UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

T E S I S

"Base de Datos MySQL con
Interfaz Java Web Services
Caso de Estudio: Recepción de Reportes de Ventas y Órdenes de Reparación de la Industria Automotriz"

QUE PARA OBTENER EL GRADO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

SAÚL MORENO MOTTE

DIRECTORA DE TESIS:

DRA. ANA MARÍA VÁZQUEZ VARGAS

CIUDAD UNIVERSITARIA
MÉXICO D.F. **2004**

Digno eres tú, Jehová, nuestro Dios mismo, de recibir la gloria y la honra y el poder, porque tú creaste todas las cosas, y a causa de tu voluntad existieron y fueron creadas. (*Una Revelación a Juan 4:11*)

A Jehová Dios, creador de la Tierra y todo lo que hay en ella.

A mis padres, Salomón Moreno Rivera y
María Teresa Motte Galicia, por darme la vida.

A mi madre, María Teresa Motte Galicia, por cuidar de mi hermano y de mí hasta el día de hoy. ¡Asegura tu cinturón que vamos a despegar!

A mi abue, Zoila Victoria Galicia Sosa quien está al tanto de mí desde mi niñez.

A mi familia, especialmente a mis tíos, quienes me invitan a ponerme metas y alcanzarlas.

A José Alfredo Motte Galicia, tío, amigo, padre de familia, maestro de la palabra de Dios, maestro de ciencias, maestro de música, maestro de álgebra lineal, maestro de cálculo vectorial, maestro de ingeniería de programación, maestro de idiomas, maestro de ajedrez, maestro de carpintería, maestro de...

A David Moreno Motte, mi hermanito, y su amada esposa, María del Carmen Rodríguez Mondragón, quienes se casaron felizmente el 22 de agosto de 2004.
¡Felicidades Mijitos Lindos!

A los fundadores de los recintos universitarios y autónomos en México,
promotores de una educación libre, profesional, imparcial y gratuita.

A mis maestros del CCH Azcapo e Ingeniería de la UNAM, por dedicar de su tiempo y energías en instruirme sobre su especialidad y enseñarme el oficio de ser estudiante.

A mis compañeros y amigos en la UNAM: Haydee, Gabriela, Arturo, Luis Alberto, René, Eduardo, César, Rodolfo, Anabel, Ivonne,…

A Ernesto Santos Lozano, Alberto Aparicio Pérez y Manuel López Nava, por compartirme su experiencia en el procesamiento electrónico de datos de la industria automotriz.

A los compañeros de la industria automotriz: Erika, Juan Carlos, Rossana, Irma, Jorge, Gabriel, Michel, Rafael, Braulio, Ramiro, MA Vallejo, Melchor, Martín, M Muñoz,…

A Agustín Arellano, maestro de idiomas, inventor de metodologías de aprendizaje.

…a todos ellos dedico éste modesto trabajo de investigación

Atte, Saúl Moreno Motte

Índice Temático

Índice de Ilustraciones

Índice de Tablas

Índice de Anexos

Prólogo

La base de datos y la interfaz de java presentados en el presente trabajo de tesis son un producto de nueva creación, lo que significa que no exponemos código ni datos propiedad de empresa automotriz alguna. No será parte de ésta argumentación el cómo resolver problemas de normalización en bases de datos en los que frecuentemente incurren los sistemas. También, nos abstenemos de explicar cómo optimizar sistemas existentes, cómo hacer modular un programa que no lo es, cómo reutilizar código, o cómo crear sofisticados mecanismos de seguridad, temas dignos de otros trabajos de tesis.

La presente investigación rescata el paradigma de los XML Web Services, a nuestro parecer muy eficaz, por ser una arquitectura que está mejorando todos los días a pasos agigantados y cuya utilización se está difundiendo vertiginosamente debido a su alta versatilidad en condiciones adversas y heterogéneas.

Daremos una buena idea sobre el uso de la novedosa tecnología que acompaña a los documentos XML, la gramática de los Schemas que, además de describir una familia de documentos, pueden involucrarse en la tarea de elaborar un Autómata Finito Determinístico automáticamente a partir de una Expresión Regular para validar un conjunto de datos. Estamos convencidos de que el caso de estudio seleccionado ilustrará el potencial de los XML Web services, metodología que todo Ingeniero en Computación dominará en poco tiempo, como hoy un adolescente de 16 años escribe y consume páginas HTML de manera cotidiana.

Moreno Motte Saúl – Tesista
Dra. Ana María Vázquez Vargas – Directora de Tesis

Introducción

1. Tema a tratar

¿Porqué ha de estudiarse un tema como "Base de datos MySQL con Interfaz Java Web Services, Caso de Estudio: Recepción de Órdenes de Reparación y Reportes de Ventas de la Industria Automotriz" en una tesis? Siento que éste título expresa muy bien 4 conceptos que en mis años de servicio en el escenario de la industria del transporte terrestre aprendí por separado para después integrar y relacionar en uno solo: bases de datos MySQL, interfases Java Web Services, Reportes de Ventas y Reportes de Órdenes de Reparación.

El tema primeramente, menciona una base de datos MySQL. Dicho elemento está compuesto de al menos tres subconjuntos funcionales, a saber, un servicio de base de datos, un motor operacional y un espacio de trabajo dedicado a almacenar un conjunto de tablas y relaciones normalizadas entre sí. Para la industria automotriz, es vital tener un soporte de datos, ya que necesita en todo tiempo, almacenar información de catálogos, entidades y datos históricos, sea cual sea la utilización de esa información.

Por otra parte encontramos en el tema la Interfaz Java Web Services. En la tesis haremos ver qué es y porqué es necesaria, y diremos qué papel juega en la solución a la problemática planteada. Además, defenderemos a los Web services sobre otras arquitecturas, dando a conocer un transfondo teórico de los sistemas heterogéneos y las diferentes alternativas que han dado empresas y organizaciones para unificar los sistemas distribuidos en una capa homogénea.

En éste punto debo advertir al lector de éste trabajo de investigación que el término "Web" hace pensar inmediatamente en el uso tradicional de Internet, y con justa razón, puesto que dichas aplicaciones han estado progresando en metodologías con el paso de los años. No obstante, construir una "Web" no es lo mismo que diseñar un "Web service". La Web define la creación de portales electrónicos informativos (estáticos o dinámicos) sobre el protocolo HTTP (incluyendo textos, HTML, archivos binarios, archivos gráficos, etc), mientras que los Web services son nuevos y especializados usos de la clásica Internet cuyas aplicaciones basadas en recientes estándares como XML le dan una nueva apariencia a la computación colaborativa.

Una advertencia adicional es que usamos indistintamente el término XML Web Services y SOAP Web services porque ésta última es una gramática particular de XML, es decir, obedece a la gramática base de lenguaje y además está delimitada por un número de reglas adicionales o "extendidas".

Finalmente, el tema describe a los reportes de ventas y reportes de reparación como el caso de estudio seleccionado para aplicar la nueva metodología. La tesis muestra la situación problemática que teníamos en la industria automotriz con el tratamiento de éste tipo de información, y cómo los Web services permitieron que sucursales del ramo automotriz enviaran de una mejor manera sus reportes de ventas y órdenes de reparación a su corporativo a la par que dieron respuesta a cada una de las necesidades de dicha situación problemática. Al mejorar el flujo y la calidad de éste tipo de información, es como se tuvo un mejor conocimiento del progreso del negocio, y de ésta manera se tuvo la forma de hacer tomas de decisiones más objetivas.

2. Los alcances y limitaciones de la investigación.

El proyecto de Web services para la industria automotriz es en realidad gigantesco, ambicioso. En la industria automotriz en la que laboré los altos directivos estaban determinados a proveer una interfaz de Web services a todo tipo de sistema, porque ello solucionaría muchos problemas de migración, incompatibilidad y duplicidad de sistemas. Las interfaces serían tanto para sistemas internos del corporativo como los sistemas para recepción de información de los distribuidores, de manera externa. De hecho, hasta se creó un departamento que diera seguimiento a dicho proyecto.

En éste sentido, el propósito en el corporativo es proveer una interfaz de Web services para todos los sistemas de los distribuidores. Aparte de los sistemas de ventas y de servicio, los directivos desean proveer interfaces de Web services para vehiculos seminuevos, vehículos al servicio de la compañía, estados financieros, facturas de unidades nuevas a distribuidores, avisos de pagos a tesorería, pedidos de refacciones, estados de cuenta de distribuidores, administración de cuentas de correo electrónico de los distribuidores, reportes de métricas de acceso a la red satelital del corporativo, etc. Por tanto, al tratarse de un trabajo tan grande, se comenzó realizando un "prototipo" para ventas y servicio. Los administradores de los departamentos de tecnología de información determinaron que si la solución funcionaba para ésta área, podría extrapolarse a las demás. Adicionalmente, se comenzó por los reportes de ventas y servicio debido a que son una necesidad apremiante, ya que empezaron a presentarse problemas de invalidez de datos, la información empezaba a llegar extemporáneamente, los distribuidores tenían que capturar la información dos veces debido a que los sistemas anteriores, no tenían interfases en línea, y otro tipo de complicaciones que trataré más adelante.

En ésta aplicación la única limitación fue el tiempo. Debido a que la industria, especialmente la automotriz, está preocupada por la educación contínua de sus recursos humanos, no tuvimos barreras técnicas o humanas que vencer. Además, en el departamento se nos proveyó todo el equipo necesario para levantar prototipos y hacer todo tipo de prueba, aún en condiciones reales con conectividad satelital real, ya que dos computadoras no se comunican igual a 1m que a 1000km de distancia. Además de ello, todas las herramientas de software que utilizamos están a disposición de cualquier usuario en Internet de forma gratuita y libre.

3. Objetivos previstos y metas reales.

Los Web services son adaptables para cualquier tipo de giro, sea de entretenimiento, comercial, educativo ó gubernamental. En términos generales, son útiles y aplicables a cualquier "mercado" donde el "sistema monetario" son los datos. Pero presentar un caso ficticio podría acarrear ausencia de credibilidad a éste trabajo de investigación. Por tanto, presentaré un caso real de la industria del transporte terrestre ¿Cuál?

En la industria automotriz ya se encuentra funcionando un conjunto de Web services para recepción de reportes de ventas y órdenes de reparación, otorgando funcionalidad a 150 distribuidores. Ésos Web services, tal como los que se exponen en ésta tesis, son la puerta a una base de datos. Obviamente, los datos recibidos en esa base tienen muchos usos, entre ellos, elaborar reportes cada hora para enviarlos a los proveedores de servicios del corporativo, a los distribuidores de la industria automotriz, y hacer consultas de métricas para el propio corporativo.

A pesar que ése Web service (al que llamaré X) está protegido por derechos de autor y políticas de privacidad, quise dar a conocer la tecnología de Web services aplicada a cualquier firma automotriz. Por tanto, en colaboración con la Doctora Ana María Vázquez Vargas y utilizando la experiencia profesional como pasante, generé un nuevo Web service, con al menos once diferencias determinantes con respecto al primero:

1.) Mientras el Web service X tiene 11 métodos, él presentado en mi tesis tiene 19.
2.) El Web service X está programado en C++ con scripts de Perl y Python, mientras que mi Web service está desarrollado enteramente en Java.
3.) El Web service X es provisto por un servidor de CGI (Apache para Linux) mientras que mi Web service es provisto por Java Web Services Developer's Package.
4.) Los mensajes del Web service X son diferentes a los que procesan mis Web services, tanto en estructura como en el nombre de los nodos.
5.) Los métodos del Web service X instalados en la industria automotriz están etiquetados en inglés, mientras que la doctora y un servidor los nombramos en español (AltaUnidad, AltaInventario,…)
6.) Los mensajes del Web service X son validados en base a programación estructurada (for, if, while, do, etc) mientras que mis mensajes son validados usando la tecnología de XML Schemas (de hecho, explicaré mas adelante qué es este concepto y porqué es tan ventajoso)
7.) El Web service X transfiere datos reales, mientras que mi Web service presentará únicamente datos ficticios

8.) El Web service X traduce los XML en llamadas a componentes de CORBA, mientras que mi Web service genera llamadas a clases de Java.
9.) La base de datos con la que interactúa el Web service X tiene algunos problemas de normalización, mientras que la base de datos de la tesis no presenta ese problema.
10.) La base de datos X cuenta con un conjunto de páginas para analizar la información, mientras que nosotros lo hacemos con el cliente versión texto de MySQL.
11.) El Web service X es de uso enteramente lucrativo, mientras que mi Web service solo tiene propósitos docentes.

Por las razones vistas anteriormente, el objetivo de éste trabajo de investigación no es hacer una réplica exacta del Web service X, poner en evidencia el trabajo de otras personas, o hacer un mal manejo de información privada. Mas bien, el objetivo general de ésta tesis es subrayar que los Web services son poderosas y sencillas herramientas de procesamiento y transferencia de información. El objetivo específico es escribir una solución completamente nueva, dando respuesta a la voz del problema original que habla sobre las necesidades de información de la industria automotriz: necesidades de un mejor conocimiento del progreso del negocio y necesidades de basar la toma de decisiones en datos confiables.

En la parte final de ésta tesis, el lector encontrará un servicio Web programado en Java, de plataforma independiente (capaz de instalarse en Linux, Windows XP o cualquier otro sistema operativo), útil para recibir en línea, de forma segura y estandarizada información de reportes de ventas y órdenes de reparación en una base de datos MySQL. Con ése conjunto de programas se pretende ilustrar propiedades y funcionalidades de arquitecturas como los Web Services, para solucionar el problema de la heterogeneidad de protocolos y lenguajes que se encuentra en los sistemas distribuidos de nuestros días.

4. El estado en el que se encontraba el tema antes de nuestra investigación.

Los reportes de ventas se enviaban usando una ventana de terminal de emulación VT3270. El problema con éste método de envío era que los distribuidores enviaban la información de ventas tardíamente, sea voluntaria o involuntariamente. Voluntariamente porque podían realizar y facturar una venta el día 15 del mes y reportarla hasta el día 28, por ejemplo, siendo que la debían reportar inmediatamente. Eso traía problemas financieros para el corporativo, sobre todo si se piensa en esos trece dias en el que la unidad es financiada por el corporativo, siendo que el importe por dicha unidad ya puede ser cubierto por la distribuidora.

Por otra parte, el corporativo solamente se enteraba de qué distribuidores no habían reportado sus ventas hasta que las oficinas centrales en Estados Unidos mandaban un reporte especial. De hecho, esta información necesita tenerla el corporativo en México antes que las oficinas centrales en Estados Unidos, para tomar acciones antes de que las buenas o malas noticias lleguen a oídos de tales oficinas centrales.

Técnicamente, los reportes de ventas también representaban un problema debido a que mantenían abierto en varios firewalls el puerto para VT3270, que es diferente para el puerto 80, lo que rebasa las políticas de seguridad de la empresa.

Similarmente, antes de la entrada de los Web services al corporativo, las órdenes de reparación se enviaban usando un servidor de ftp, con un cliente provisto por el corporativo. Esto representaba un problema en el sentido de que una vez que el archivo es recibido por el servidor, podría llegar a haber forma de validarlo pero no de divulgar los resultados de esa validación. La utilidad de la validación viene desde el momento que se informa inmediatamente al usuario que envía esa información que sus datos no cumplen siquiera ni con los requisitos mínimos.

El uso de los Web services se ve justificado al momento de darse cuenta de que los problemas y las necesidades supracitados se ven abatidos con tal metodología.

5. La metodología utilizada.

La argumentación abre con una exposición detallada de los rasgos determinantes de la situación problemática. Dicha problemática pudo haber sido tratada usando tres tipos de arquitectura, RMI, CORBA y Web Services. El primer paso fue investigar tales arquitecturas por separado, colocar las tres sobre la mesa, seleccionar una de las tres y justificar el porqué de dicha selección.

Seleccionar un motor de base de datos, un método de validación y un lenguaje de programación no fue tarea difícil, pero lo que representa un problema es asegurarse de que se esté utilizando la arquitectura de sistemas distribuidos correcta. Es apremiante encontrar la mejor alternativa porque una selección inadecuada de herramientas y tecnologías en la solución de éste tipo de problemas redunda en costosas pérdidas de información.

Algo útil que encontrará el lector en éste trabajo es la metodología para validar XML a través de XML Schemas, y la manera de construir Autómatas Finitos Determinísticos a partir de expresiones regulares para validación de datos.

6. Estructura del trabajo.

Mi tesis está dividida en seis capítulos. En el capítulo 1 expongo qué problema teníamos por resolver, y describo el escenario en el que se desarrolla el problema. Además de reconocer diez barreras técnicas a vencer, defino las reglas de negocios del corporativo sobre el tratamiento de la información de los reportes de ventas y las órdenes de reparación. De hecho, en éste capítulo visualizamos que el problema requiere una metodología de componentes distribuidos segura, eficaz, abierta y confiable, sin llegar a especificar qué metodología era esa. Al dejar planteado el problema, el siguiente paso es buscar alternativas de solución.

El capítulo 2 está enteramente dedicado a escoger un conjunto de metodologías y herramientas para tratar con el problema. En éste capítulo damos los argumentos que nos impulsaron a escoger a Java 2 como nuestro lenguaje de programación, a MySQL como nuestro motor de bases de datos relacionales y a las expresiones regulares y autómatas finitos determinísticos como nuestra metodología de validación de tipos de datos. Si bien apostar a estos elementos no fue tarea que redundara en complicación alguna, apareció ante nuestros ojos la necesidad de escoger una arquitectura de sistemas distribuidos, debido a que nuestro problema se hallaba imbuido en un escenario de componentes heterogéneos y en red.

Por tanto, nuestro cuarto As sería un modelo de objetos distribuidos, pero ¿Cuál? Debido a que Java 2 contiene tres de ellos –CORBA, RMI y Web services-, nos pareció adecuado estudiar a cada uno de ellos por separado y hasta cierto nivel de detallle, para después compararlos característica por característica. Con ello, tomamos nuestra decisión de una forma más racional y justificable. De hecho, ésta pequeña travesía nos sirvió para conocer un poco más sobre los conceptos del cómputo distribuido de nuestros tiempos y esperamos que al lector también le beneficie el capítulo 2 en éste sentido.

Una vez que seleccionamos la arquitectura de Web services para resolver nuestro problema particular, nos dimos a la tarea de investigar a profundidad cada concepto involucrado en ésta metodología. Por tanto, el capítulo tres separa dichos elementos y es como estudiamos por separado el protocolo HTTP orientado a servicios y el formato XML orientado a comunicaciones entre aplicaciones, desdoblándose en las gramáticas XML Schemas, SOAP y WSDL. En éste capítulo el lector comprenderá que cada gramática tiene un uso particular e intensivo en el desarrollo y explotación de los Web services.

En la última sección del capitulo 3 reuniremos todos los conceptos de Web services para que el lector perciba cómo todos los elementos están íntimamente involucrados en una arquitectura orientada a servicios.

Al llegar a este punto, reunimos todos los elementos necesarios para crear la solución a la problemática original, y las herramientas y metodologías que usaríamos serían Java, MySQL, los Web services de Java, y las múltiples bibliotecas de Java al servicio de nuestro desarrollo. Con éstos elementos en mente nos comprometimos a crear una solución al requerimiento de recepción de órdenes de reparación y reportes de ventas de la industria automotriz. Por lo anterior, en el capítulo 4 se da una propuesta de solución, dividiéndose

en un diseño de base de datos y la utilización de Java para recibir, validar y procesar XML, al mismo tiempo que probamos el servicio creando un pequeño cliente en MSAccess, con una biblioteca simple conocida como MSSOAP Toolkit. Concluimos el capítulo 4 hablando de los posibles mensajes de error a los que se enfrenta el distribuidor y en términos generales, cualquier usuario de nuestra interfaz de Web services hacia la base de datos corporativa.

Después de pensar en el problema, proponer una solución y condensarla en un conjunto de programas y documentos XML, nos preguntamos si habían sido abatidas las características que deseabamos modificar del problema original. Por tanto, hicimos una revisión de cada uno de los diez retos técnicos referidos en la hipótesis, para confirmar que la solución dada extinguiera íntegramente la situación adversa a la que nos enfrentamos. En éste sentido, el capítulo 5 aborda los resultados positivos que descubrimos al introducir éstas prácticas de negocios entre los distribuidores y sus proveedores de sistemas certificados. El capítulo 6 resume ésta lista de necesidades y satisfactores, y da a conocer la mejoría en el conocimiento de las variables críticas de la industria automotriz. Al tener éste último, los resultados en la empresa son, obviamente, llevar a cabo mejores acciones correctivas y preventivas, pensando en el siguiente reto de crear un proceso realmente estable.

Esperamos que, después de haber revisado éste material, el Ingeniero en Computación llegue a ver la Web desde ótra óptica, para que recuerde que las aplicaciones de Internet han sido trasladadas fuera del navegador casero al cual estuvieron esclavizadas por muchos años. También deseamos que ésta investigación sea una buena referencia de servicios Web y un punto de partida para posteriores investigaciones.

Moreno Motte Saúl

1 Planteamiento del problema a resolver

Laboré poco más de cuatro años para una firma de la industria automotriz de rango global del 4 de octubre de 1999 al 12 de enero de 2004. Me situé en una sucursal corporativa de Mexico, que tiene a su cargo a distribuidores de marca a nivel nacional, a la par que es dependiente de las oficinas centrales cuya sede está en Estados Unidos de Norteamérica[1]. Me inicié en el área de "Enlace con el Distribuidor" como ingeniero de soporte en materia de aplicaciones corporatívas, así como sistemas hechos en el mismo departamento para captura, procesamiento y transferencia del distribuidor al corporativo, que conocíamos como "inhouse".

Durante el último par de años en este lapso de tiempo el giro del departamento cambió radicalmente. Tanto los sistemas propios del distribuidor como los sistemas "inhouse" desaparecieron y en su lugar quedaron productos de software de diversos proveedores. El corporativo promovió la libre competencia por éste nicho mercantil. En tal mercado, todos los vendedores de programas ahora harían la tarea de los sistemas inhouse y las tareas informáticas internas de la distribuidora – ventas, servicio, refacciones, contabilidad, nómina, etc-

Debido a que los productos de software de dichos proveedores tuvieron que ser sometidos a un minucioso proceso de certificación, los conocíamos como "los sistemas certificados".

¿En torno a qué criterios fueron calificados dichos sistemas? Dichos sistemas fueron evaluados en base a las siguientes categorías:

1. **Perfil técnico**. El corporativo evaluó los recursos de hardware, software y comunicaciones que el sistema requiere. Además, se auditó la estructura de datos que soporta la lógica del negocio, y cómo se da mantenimiento a base de datos. También fue importante para el corporativo conocer qué herramientas de desarrollo usó el proveedor, qué documentación entrega, qué tan natural, y qué tan seguro es manejar el sistema, cuales son los mecanismos de auditoria y manejo de errores y finalmente qué pruebas de software se efectuaron sobre dichos productos.
2. **Soporte**. El corporativo calificó la integración, compatibilidad, soporte técnico, infraestructura, capacitación, entrenamiento y esquema para medición de satisfacción del cliente maneja el proveedor del producto requerido para trabajar en la industria automotriz.
3. **Condiciones comerciales**. El corporativo preguntó acerca de cómo maneja los contratos el proveedor con los distribuidores, y qué métodos de cotización hay en el sistema.
4. **Implantación**. El corporativo indagó en la metodología, documentación, implantación y enfoques de instalación del producto en el distribuidor.
5. **Lineamientos de autos nuevos**. El corporativo revisó que el producto del proveedor tuviera un razonable esquema de marketing para ventas de autos nuevos en adquisiciones, inventarios, comisiones y facturación.
6. **Lineamientos de refacciones**. El corporativo también exigió que el producto del proveedor manejara un esquema de marketing para refacciones en adquisiciones, inventarios, comisiones y facturación.
7. **Servicio**. El corporativo solicitó que el proveedor demostrara que su sistema es apto para tratar información del taller tocante a ventas, servicio, inventarios, facturación, administración de clientes, asesoramiento de la imagen del vehículo y cuidado de la calidad en el trabajo.
8. **Administración del distribuidor**. El corporativo evaluó que el sistema tuviera el soporte de datos adecuado para manejar información de caja, bancos, compras, contabilidad, cuentas por cobrar, cuentas por pagar, crédito, cobranzas y nóminas.
9. **Lineamientos corporativos**. Un concepto que está de moda no por lo bien que luce sino por el dinero que atrae a los negocios es CRM, o bien "Customer Relationship Management", que se encarga de estar en contacto constante

[1] No mencionaré nombres de marcas, ni modelos, ni personas, ni procedimientos internos, ni localidades geográficas exactas, -en la medida de lo posible- para evitar cualquier tipo de suspicacias en torno a derechos reservados, derechos de autor o políticas de privacidad. Tampoco estoy obligado a promover la preferencia de una firma automotriz sobre otra en mi sustentación de argumentos estrictamente académica, sin efectos lucrativos.

con el cliente, para seguir de cerca el desempeño de la unidad, y el nivel de satisfacción del cliente. Por definición, un cliente satisfecho con un servicio es un cliente que sabe donde encontrar lo que necesita, y ese lugar es la distribuidora automotriz. Adicionalmente a esto, el sistema debe manejar internamente prospectos de venta, porque no hay nada mejor que conocer información de clientes potenciales para eventualmente invitarlos a incorporarse a algún plan de venta a crédito, arrendamiento, o descuentos especiales.

Durante el proceso de certificación, los niveles directivos se dieron cuenta que los sistemas certificados, además de manejar consistentemente los datos del concesionario, debían tener la capacidad de enviar información a los sistemas corporativos, los cuales permanecieron sin cambios.

Dentro de los sistemas corporativos que manejamos en aquel tiempo se encuentran:

1. **Los reportes de ventas de vehiculos nuevos del distribuidor al corporativo**. Esa información tiene origen en vehículos tanto de importación de Brasil, Argentina, Gran Bretaña y Estados Unidos, como de producción nacional. Los datos también tienen su origen en modelos de otras marcas en el consorcio. Estos datos son base para hacer el conteo en dias del financiamiento que otorga la marca, así como el financiamiento que otorga las instituciones de crédito asociadas. La información que recibe este sistema también es utilizada para llevar el control de garantías y campañas de servicio sin costo al distribuidor.
2. **Los reportes de servicio hechos a unidades seminuevas del distribuidor al corporativo**. Esta información es esencial para obtener el punto de vista del cliente acerca del servicio que le fue otorgado en el taller de la agencia, ademas de saber cuanto tiempo pasó la unidad del cliente en la agencia.
3. **Rastreo de vehículos seminuevos en el corporativo**. El corporativo lleva un registro actualizado de unidades que fueron utilizadas por la compañía, que recibieron mantenimiento en el taller del corporativo, y que han sido publicadas aleatoriamente para que los empleados puedan adquirirlas en venta de contado, sin importar la planta o patio de origen.
4. **Registro de vehículos al servicio de la compañía a nivel nacional de corporativos**. Son vehículos nuevos en poder de los altos directivos de la compañía, unidades que mas tarde se convertirán en vehículos seminuevos.
5. **Envío de estados financieros del distribuidor al corporativo, y del corporativo a la sede**. Los departamentos de administración de negocios de las entidades mencionadas requieren recibir esta información de todos los distribuidores de forma mensual.
6. **Envío de facturas de unidades nuevas del corporativo a los distribuidores**.
7. **Notificación de pagos a tesorería del distribuidor al corporativo.** El corporativo debe cerciorarse que todos los distribuidores estén haciendo periódicamente pagos a la SHCP.
8. **Envío de pedidos de refacciones del distribuidor al corporativo**. Los dos tipos de pedidos son “express” o de emergencia y “stock” o de reserva.
9. **Rastreo de pedidos de refacciones del distribuidor**. La gerencia de refacciones del distribuidor requiere saber continuamente el estatus de cada linea de pedido solicitada, para saber qué piezas se surtirán, que material quedará pendiente para el siguiente día y que líneas quedarán definitivamente canceladas.
10. **Solicitud de precios de refacciones en linea al corporativo**. Esta información corporativa elimina los boletines y catálogos de partes con precios desactualizados. El distribuidor debe tener una lista actualizada de todos sus precios, incluyendo precio público, precio mayoreo y precio distribudor). También debe conocer que lineas sufrieron altas, bajas y cambios.
11. **Solicitud de estados de cuenta de distribuidores**. El distribuidor desea conocer el límite de su línea de crédito con el corporatvo, así como sus deudas y los conceptos de deuda, tal como un tarjetahabiente consulta su saldo en Internet.
12. **Administración de cuentas de correo electrónico para los distribuidores**.
13. **Reportes de métricas para el acceso a la red satelital del soporte técnico al corporativo sobre cada una de esas operaciones.**
14. **otros**.

Ya que el departamento de enlace debía proveer una interfaz para recibir toda esta información y en vista de que se tienen múltiples sistemas corporativos, **la idea sería llevada a la práctica con un prototipo que incluyera solamente** (1) **Los reportes de ventas de vehiculos nuevos** y (2) **Los reportes de servicio hechos a unidades seminuevas en el taller del distribuidor.**

En vista de que nuestro problema estaba adscrito a éstos dos tipos de reportes, empezamos a reunir más información del asunto. Los actores involucrados en este escenario son el Cliente Final, los Distribuidores, el Proveedor de Encuestas al Cliente final, el Corporativo Nacional, Las plantas Nacionales e Importaciones, y las Oficinas Centrales en Estados Unidos. Al tratar de comprender como trabajan las partes e interrogando a las personas correspondientes dentro de la compañía encontramos que el corporativo necesita conocer el progreso del negocio y la salud del mismo, a través de variables críticas.

La primera variable crítica es la venta global de unidades nuevas a los clientes finales por día así como la venta mensual por distribuidor –que conocíamos como "venta de menudeo"-[2]. Esta variable es crítica porque en base a ella se generan órdenes de producción, pedimentos de importación, campañas de ventas –meses de descuentos, ferias de liquidaciones, precios de introducción de nuevas líneas, etc.-, operaciones de financiamiento, operaciones de garantías y demás procesos internos.

Hablando acerca de las ventas de menudeo, el corporativo necesita conocer al menos qué vehiculo fue vendido, a quién fue vendido y la fecha de venta.

- **Vehiculo.** Necesita saber qué vehiculo fue vendido para reponer el inventario del corporativo, y eso solo se puede hacer con producción e importaciones. En tal caso, el distribuidor, concesionario, filial o dueño de franquicia, como deba llamársele, deberá enviar el número de serie de vehiculo, número de identificación "vin", o como se desee nombrarlo, porque a partir de esta llave el corporativo puede saber exactamente el año, el modelo, la carrocería, el motor y el catálogo del vehiculo. Todo en un solo paquete de unos cuantos bytes.
- **Cliente.** Es necesario saber a quién fue vendido para conocer el punto de vista del cliente acerca de la agencia, del trato que tuvo por parte del personal, y de la calidad del producto que adquirió con su valioso salario. Esto implica que se debe tener al menos un teléfono para poder localizar al cliente, sea en su casa, oficina, celular, etc. Entre más teléfonos se tengan del cliente es mejor, porque así existen más probabilidades de localizarlo y encuestarlo.
- **Fecha de venta**. Ésta delimita el tiempo que se dará de garantía para que el cliente pueda hacer una reclamación por desperfecto de la unidad o solicitar servicios sin cargos adicionales para su unidad en la agencia. Además este último punto tiene implicaciones monetarias, porque generalmente, para toda industria de giro automotriz, el corporativo extiende las unidades a crédito en las distribuidoras. El saber que esa unidad ya fue vendida al cliente final, implica que el distribuidor ya puede empezar a pagar la unidad al corporativo en una o varias exposiciones, sea que el cliente final de la unidad haya pagado de contado o esté haciendo uso de los servicios de una o varias instituciones de crédito. Como comentario adicional se puede decir que si un distribuidor reporta ventas, pero no hace los pagos a la cuenta corporativa, a la larga sufrirá un bloqueo en su línea de crédito y no podrá solicitar más unidades nuevas. Es cierto el dicho que "la casa no pierde" y aplica en el caso del corporativo.

Una situación problemática adicional relacionada con las ventas de menudeo residía en el desconocimiento del inventario real del distribuidor. Dicho inventario debe estar actualizado no solo para con el distribuidor, sino para con el corporativo y ser en todo momento del conocimiento del corporativo. Si bien es cierto que el corporativo sabe con exactitud cuantas unidades le vendió al distribuidor por mayoreo, desde luego que se desconoce qué unidades tiene el distribuidor en su inventario en un momento determinado, debido a los movimientos diarios con clientes (venta y devolución) y con distribuidores (transferencia e intercambio). Se necesitan esos datos en un repositorio, para que los consumidores de unidades nuevas –reales y potenciales- los puedan consultar, con el fin de saber qué unidades están disponibles en su agencia más cercana[3].

Un problema más es que no existía un registro de las transferencias e intercambios de unidades entre distribuidores

Al seguir investigando sobre el tema de ventas de menudeo con el personal correspondiente, se formularon las siguientes reglas de negocios:

1. El sistema estará soportado por una base de datos relacional cuyas entidades son las Agencias, los Clientes y los Vendedores de Unidades. La base de datos deberá utilizar conjuntos de datos finitos para conservar la ortogonalidad de los datos de Estados, Tipos de Venta, Tipos de Pago, Claves del Cliente, Tipos de Teléfono y

[2]La "venta de menudeo" es la venta del distribuidor al consumidor final, y la "venta de mayoreo" es la venta del corporativo al distribuidor, usualmente a crédito con 50 días sin intereses de financiamiento.
[3]La información de inventarios alimenta a otro sistema –el sistema de ventas en línea por Internet en apartado con tarjeta de crédito-.

Teléfonos. Además la base de datos deberá registrar los movimientos históricos de Ventas, Transferencias e Inventarios.

2. El sistema aceptará los datos del cliente solo si se reportan uno o más teléfonos (de casa, de oficina o celulares).
3. Se podrá reportar una venta si el comprador y/o conductor están en la tabla de cliente.
4. Se podrá reportar una venta si el vehiculo está en el inventario de la agencia.
5. Si se reportan m unidades en el inventario, y n unidades ya existen en la tabla de inventario, entonces se insertarán m-n unidades en la transacción, y dicha transacción se reportará como exitosa.
6. Si una agencia reporta una unidad como vendida, entonces la única operación que podrá efectuar sobre dicha unidad es una cancelación de venta
7. Si una agencia transfiere una unidad al inventario de otra agencia, entonces la única operación que podrá efectuar sobre dicha unidad es una autotransferencia.
8. Las transferencias y autotransferencias de unidades son legales solo si ambas agencias han convenido en ello (en muchos casos se da el intercambio de unidades entre agencias y físicamente, la agencia A envía su chofer a la agencia B con un carro, y se regresa con el carro de intercambio y con la factura impresa).
9. Se podrá realizar la autotransferencia de una unidad si dicha unidad todavía no ha sido reportada por otra agencia como vendida. En este caso se notificará al distribuidor afectado con un mensaje de correo electrónico o algún otro medio disponible en el sistema.
10. Si una operación de autotransferencia falla, se deberá solicitar al distribuidor que mandó el reporte de la venta que genere una cancelación de venta en dicha unidad.
11. Al cancelar una venta, ésta se dará de baja de la tabla de ventas y se regresará al inventario del distribuidor.
12. Cada treinta días el sistema vaciará automáticamente las unidades que no han sido reportadas como vendidas de la tabla de inventario, lo que obligará a cada distribuidor a enviar su inventario físico con el método Web services correspondiente.
13. Para hacer el procedimiento de transferencia, deberán revisarse las condiciones necesarias y suficientes para éste tipo de operación, tales como existencia de la unidad en el inventario, el que la unidad no esté reportada comovendida, y que los movimientos en la futura tabla inventario no sean erróneos , de acuerdo con un diagrama de flujo como el siguiente:

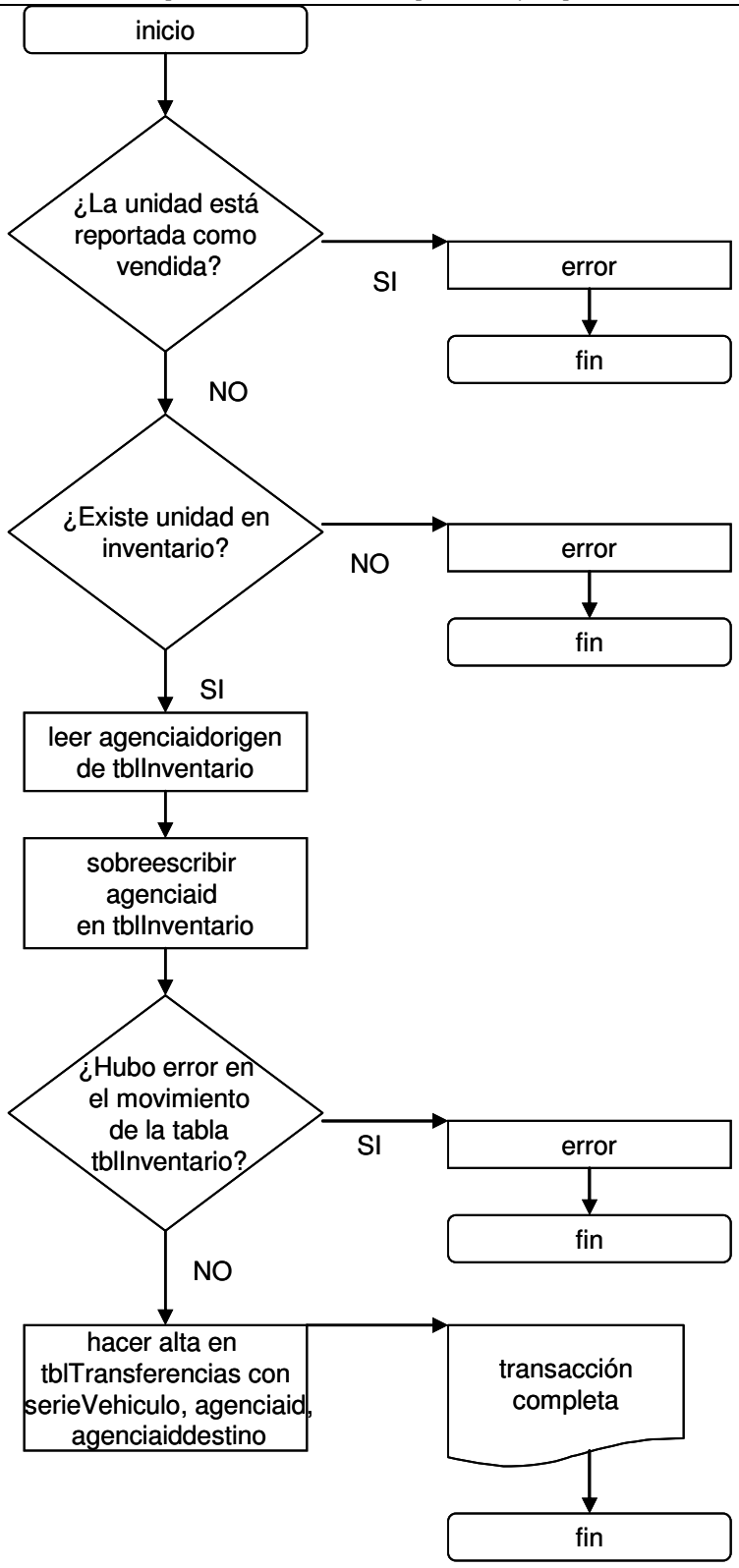


Ilustración 1 Diagrama de flujo de transacciones

Por otra parte, la segunda variable crítica a conocer es las órdenes de reparación por día, por semana y por mes. Esta variable también es crítica porque el corporativo necesita asegurarse que el cliente final esté recibiendo un servicio de calidad para su unidad, ya que una de las filosofías –o políticas- de la industria automotriz es mantener al cliente cautivo a través de su lealtad a la marca. Es decir, un cliente satisfecho pensará en renovar su unidad con la marca –a mediano y largo plazo- siempre y cuando el producto y el servicio cumplan y en la medida de lo posible, sobrepasen sus expectativas, no importando si el precio es un poco elevado en relación con otras marcas competidoras. De hecho, el cliente satisfecho no dudará en invertir en una nueva unidad más reciente, del mismo modelo o de un model similar al de su preferencia. Por ello, la política de las empresas automotrices es vende modelos 2005 en 2004, vender modelos 2006 en 2005 para siempre pensar positivamente en ventas a futuro.

Hablando acerca de las ordenes, el corporativo necesita conocer qué vehiculo fue reparado, el tipo de reparación, a quién se dio el servicio y cuando se abrió una orden y cuando se cerró.

- **Vehiculo.** Necesita saber qué vehiculo fue reparado. En este caso también se debe enviar el número de serie de vehiculo o vin.
- **Tipo de reparación**. No es lo mismo recibir notificación de que se está otorgando un servicio "de rutina" –esto es, de los 20,000 km o 40,000 km-, a recibir notificación de ajustes o reemplazos por defecto de fábrica. Al llevar el corporativo un control estadístico sobre la frecuencia de estos casos, puede detectar áreas de oportunidad en sus plantas ensambladoras. Ha habido casos que a una completa línea de vehículos ha tenido que cambiársele el juego de tornillos de la antena de am/fm, un desperfecto en el tablero, o en las vestiduras. ¡Pero ha habido casos más graves en el que ha tenido que reemplazarse la dirección, la transmisión, la caja de velocidades e incluso el motor entero! En estos casos que corre peligro la imagen de la compañía, se lanzan campañas de servicio sin costo alguno para el cliente, casos que se procuran atender a la brevedad.
- **Cliente**. De nueva cuenta, es necesario saber a quién se dio el servicio para conocer el punto de vista del cliente acerca de la agencia, del trato que tuvo por parte del personal, y de la calidad y precio del servicio adquirido. Esto implica que se debe tener al menos un teléfono para poder localizar al cliente, sea en su casa, oficina, celular, etc. Entre más teléfonos se tengan del cliente es mejor. En años recientes toda la industria ha concordado en que lo más importante para hacer negocios es el cliente, y una de las metodologías para mantenerlo cautivo es darle un producto y muchos servicios de alta calidad.
- **Fecha**. También es indispensable conocer el rango de fecha en el que la unidad permaneció en la agencia. La idea no es que un vehiculo esté atrapado en una agencia por días –o semanas-, sino que ande circulando por la calle con un feliz consumidor manejándolo desde dentro –el automóvil es la herramienta de trabajo de casi todas las empresas-.

Para el sistema de órdenes de reparación, también se recopilaron algunas reglas de negocios:

1. El sistema estará soportado por una base de datos relacional cuyas entidades son las Agencias, los Clientes y los Asesores de Servicio. La base de datos deberá utilizar conjuntos de datos finitos para conservar la ortogonalidad de los datos de Estados, Claves del Cliente, Tipos de Teléfono, Teléfonos, Tipo de operaciones, Tipo de órdenes y Tipo de servicios. Además la base de datos deberá registrar los movimientos históricos de Órdenes de Reparación.
2. El sistema aceptará los datos del cliente solo si se reportan uno o más teléfonos (de casa, de oficina o celulares)
3. Se podrá reportar una orden si el cliente está en la tabla correspondiente.
4. Se podrá reportar una orden de reparación aunque el vehículo no esté en el inventario del distribuidor.
5. Las órdenes con fechas de cierre futuras serán discriminadas en el conteo para los objetivos mensuales. Esto es, si una orden se reporta el día primero de diciembre de 2005 con fecha de cierre del primero de enero de 2006, esa órden será discriminada y no será tomada en cuenta hasta el primero de enero de 2006 si no se realiza la corrección correspondiente en ella.
6. Las órdenes podrán tener opcionalmente valores de ingresos y utilidad por conceptos de refacciones y mano de obra en hojalatería y pintura o en taller.
7. Las órdenes que no se envían con fecha de cierre serán consideradas como abiertas. La fecha de apertura es obligatoria para todas las órdenes de reparación.

Una órden de reparación puede tener dos estados. Se dice que está abierta cuando se ha reportado sin fecha de cierre, lo que le da a entender al corporativo que la unidad aún se encuentra en taller o en hojalatería y pintura de la distribuidora. La órden está cerrada cuando la unidad ha abandonado ya la agencia, y se ha levantado la factura correspondiente de esa unidad. De hecho, las órdenes que cuentan son las cerradas, y si una unidad está abierta en la fecha de corte o a final de mes, se tomará para el siguiente mes, hasta que esté cerrada.

La información de órdenes de reparación cerradas se envía a un proveedor de estadísticas externo, quien se encarga de hacer las encuestas a los destinatarios finales de productos/servicios. Si el comentario del cliente es favorable, la órden no solo es real, sino que además cuenta para el objetivo de órdenes mensual. De hecho, las calificaciones se regresan de nueva cuenta al corporativo nacional.

¿Qué objetivo tiene contar reportes de ventas y reportes de órdenes de reparación? Pues bien, si los objetivos son alcanzados –o sobrepasados-, se otorgan incentivos económicos a la distribuidora, como bonos de productividad por haber alcanzado sus objetivos mensuales. Si los objetivos no son alcanzados puede haber sanciones, desde las más ligeras hasta las más graves, es decir, no recibir incentivos, ser penalizado con multas monetarias en la cuenta del distribuidor, y en casos graves, perder permanentemente la franquicia o la certificación del corporativo nacional.

La recepción de reportes de ventas y de órdenes de reparación ya se venía haciendo desde antes del requerimiento, pero conllevaba determinados problemas técnicos. ¿Qué problemas son esos? Al menos se pueden citar diez:

1. **Información extemporánea**. Tanto los reportes de órdenes de reparación en taller como los reportes de ventas de unidades nuevas llegaban tardíamente en el mejor de los casos y fuera de tiempo en el peor de los casos.
2. **Recaptura de órdenes de reparación en taller y reportes de ventas de unidades nuevas**. El distribuidor, después de capturar éste tipo de información en sus sistemas propios, debía hacerlo en los corporativos, sin tener un medio común para compartir la información. De hecho, ése era el pretexto de enviar tardíamente la información.
3. **Información inválida**.
    a. La información se recibía en el corporativo sin la validación básica por deficiencias en los sistemas del distribuidor
    b. Por otra parte, en las agencias distribuidoras de automóviles, se tiene alta rotación de personal en ventas, servicio, y todas las áreas. Tales eventos conllevan el problema persistente de que el personal recién contratado tiende a enviar datos inválidos debido a su inexperiencia en el procesamiento y transferencia de información.
    c. El no establecer algún punto de revisión de datos de los reportes de ventas y servicio ocasionaba una detección tardía de errores. Tanto los proveedores de encuestas como la sede en Estados Unidos, rechazaban datos inválidos de los distribuidores, siendo que el corporativo podía evitar tal problema.
    d. Alto índice de información errónea por distribuidor. Este hecho desemboca en que el distribuidor no alcanza sus objetvos de ventas u ordenes porque al cliente le faltaron teléfonos, porque el numero de serie de la unidad venía mal escrito, porque la dirección del cliente era errónea, y muchas otros detalles evitables.
4. **Brecha de seguridad en los firewalls.**
    a. Los reportes de ventas de unidades nuevas eran capturados usando una terminal de emulación VT3270 hacia un equipo AS400, hecho que abría puertos no permitidos en los firewalls nacionales y norteamericanos, ocasionando un descontento para los administradores de dichos equipos por ser una amenaza potencial.
    b. Los archivos de órdenes de reparación en taller eran **transferidos** a través de un cliente y un servidor de ftp, lo que también abre una brecha en los firewalls por los puertos 20 y 21.
5. **Ausencia de descripción en los datos**. Para las órdenes de reparación, la información transferida se encontraba en formato plano o "flat ASCII". Interpretar tales archivos requeria un "layout" por separado, cosa que era riesgosa porque si el layout cambiaba al agregarse o eliminarse longitudes, encabezados, pies de página, etc, el archivo ASCII no daba señales visuales de ello.
6. **Infraestructura de soporte de costo elevado** Las órdenes de reparación se recibían en un servidor FTP del corporativo. Los primitivos servidores de éste protocolo, como las Digital MicroVax 3300 con Multinet, tenían la ventaja de ser contactados también a través de la vía telefónica con el protocolo BLAST, pero debido a que el tiempo vuelve obsoleta a cualquier computadora, éste servidor adquirió la característica de ser de caros mantenimientos preventivos y correctivos.

7. **Diversidad de infraestructuras en los distribuidores**.
    a. Los sistemas operativos del distribuidor en ocasiones no eran compatibles con los sistemas para transferir información por ftp, ni para usar la terminal de emulación VT3270.
    b. La diversidad de proveedores de software provoca que se tenga un sistema heterogéneo.
8. Si bien se conocía parcialmente el manejo de reportes de ventas y órdenes de reparación, se **desconocía el manejo de información** entre distribuidores tal como:
    a. No se estaba manejando el Inventario de unidades nuevas de distribuidor, información necesaria para publicarse en internet
    b. **Ni las transferencias de vehiculos, ni los intercambios** de una distribuidora a otra se estaban registrando en alguna parte.
9. **Se estaban desaprovechando los protocolos de transporte conocidos (HTTP), y nuevos protocolos de codificación (XML). Además se estaban desperdiciando las nuevas tecnologías de información y los modelos de objetos distribuidos.**
10. **Se tenía un bajo volumen de datos recibidos por distribuidor.** Este hecho también provoca que el distribuidor no alcance sus objetvos de ventas u órdenes por ausencia de información.

Gráficamente, ¿Qué relación existe entre los elementos del problema? A continuación se muestra el escenario que existía en ese momento en un sencillo diagrama de estereotipos:

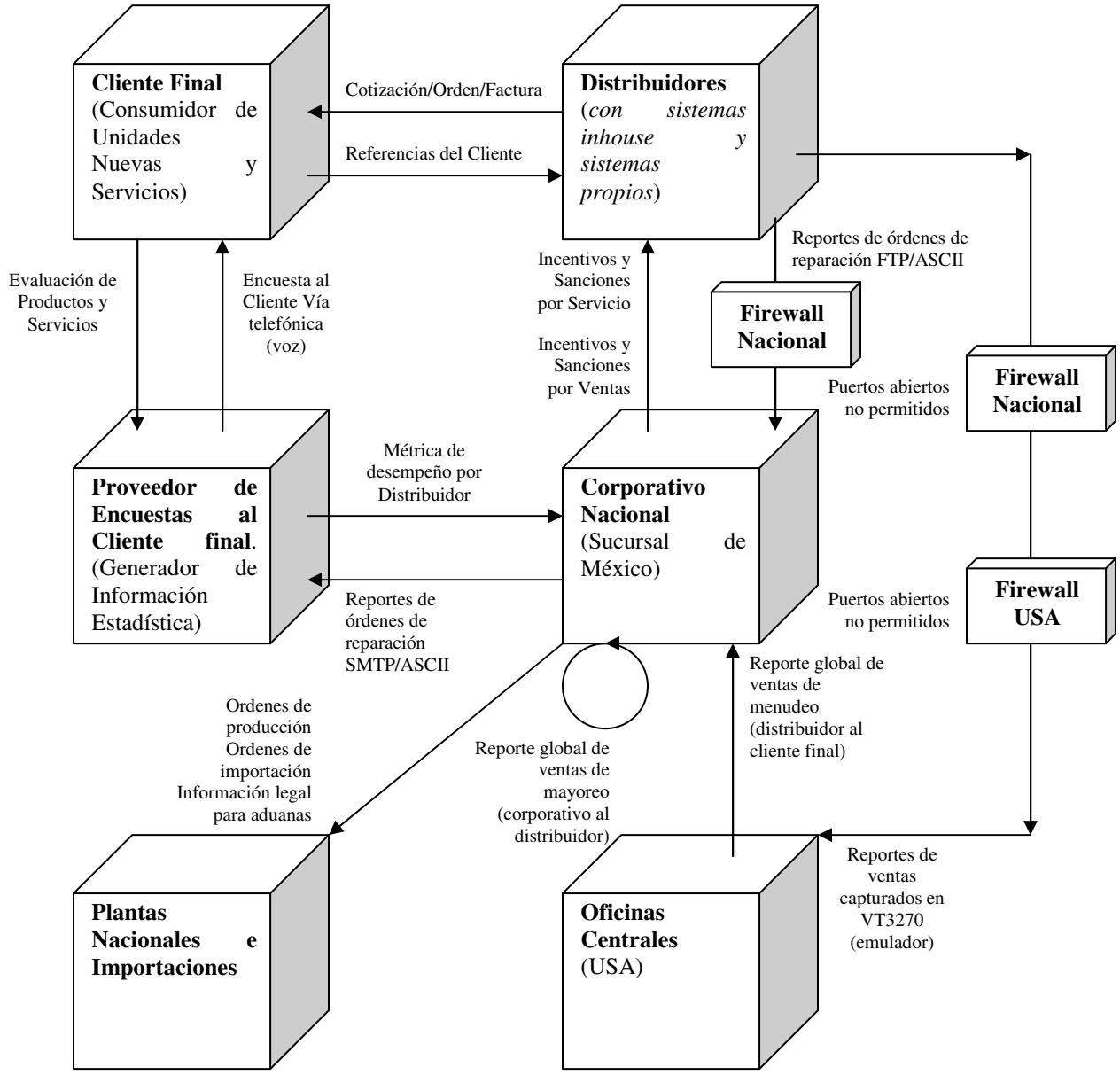


Ilustración 2 Escenario del problema

Para dar respuesta a los requerimientos y resolver este conjunto de problemas es necesario seleccionar los métodos e instrumentos metodológicos más adecuados, estudiar el método seleccionado y plantear una hipótesis que determine un camino a seguir, los pasos necesarios y las herramientas auxiliares para solucionar este problema de integración de datos de la empresa. De ello tratarán los siguientes capítulos.

2 Selección de Instrumentos metodológicos para ubicar y sistematizar el problema

2.1 Lenguaje de programación: Java 2 Enterprise Edition

¿Por qué se eligió a Java para programar la aplicación? Java es un lenguaje que tiene características llave que lo hacen idóneo para dicha tarea. Todas sus características son ya bien conocidas por la comunidad de desarrolladores de software, y las más importantes son:

1. Java es un lenguaje **orientado a objetos**. Sus librerías de clases también están orientadas a objetos.
2. Java usa una máquina virtual para ejecutar sus programas, lo que proporciona portabilidad en múltiples plataformas.
3. Java es más **fácil de usar** con respecto a otros lenguajes debido a su avanzada tecnología incorporada en sus clases.
4. Java es **dinámico y distribuido**, es decir, las clases de Java pueden ser descargadas dinámicamente sobre la red según sea requerido. Además, Java provee un amplio soporte para programación cliente servidor y programación distribuida. Tiene incorporado en esta característica un compilador "Just In Time".
5. Java es **multiproceso**. Los programas de Java pueden contener múltiples hilos para efectuar diversas tareas en paralelo. La capacidad de multihilos está construida dentro de Java y está bajo control de la JVM, no del sistema operativo.
6. Java es **robusto y seguro**. Java tiene capacidades preconstruidas para prevenir la corrupción de la memoria. Java maneja automáticamente los procesos de direccionamiento de memoria y revisión de los límites en los arreglos. Prohibe aritmética de a puntadores, y restringe a los objetos a espacios etiquetados en la memoria.
7. Los programas de java pueden adquirir varias formas según su **área de aplicación**: pueden ser "applets", "servlets" o "applications".
8. Los programas en java pueden estar diseñados como **componentes almacenados** en el servidor que forman aplicaciones escalables de Internet. El modelo actualmente aceptado para el cómputo de Java en Internet divide el proceso de aplicacón en algunas capas lógicas. Para utilizar este modelo, Sun definió Java 2 Enterprise Edition (J2EE). Existen cuatro capas lógicas:
    i. **Capa del cliente**. Cuando Java necesita ejecutarse en máquinas cliente, es típicamente implementado como un applet basado en el navegador. Este cliente ligero puede ser una página Web enviada desde el servidor en HTML.
    ii. **Capa de Presentación**. Esta se ejecuta en el Web Server. El codigo en esta capa maneja la presentación de la aplicación en el cliente. Las características comunes de Java para esta función son los servlets y las JavaServer pages (JSPs). Tanto los servlets como las java Server pages pueden generar HTML dinamico para mostrar las páginas Web a los clientes.
    iii. **Capa de aplicación**. Java puede ser utilizado en un servidor de aplicaciones para crear lógica de negocios reutilizable y publicable, en forma de componentes de aplicación. Los modelos comunes para esto son los Enterprise JavaBeans (EJB) y CORBA.
    iv. **Capa de datos**. Muchos servidores de datos ahora empiezan a proveer no solo almacenamiento de código java, sino también su ejecución, particularmente en el caso de que el código maneja una abundante cantidad de datos, o para obligar a que se cumplan con reglas de validación en los datos.
9. Java provee los elementos necesarios para programar una aplicación distribuida, ya sea usando la arquitectura CORBA, RMI, o Web services[4].

[4] CLEMENTS, Nick, et al., Introduction to Java: Java Programming, pp. 3-13

2.2 Base de datos relacional: MySQL

¿Por qué se decidió por escribir la base de datos de este sistema en el motor MySQL? Esto es debido a que dicha herramienta cuenta con los siguientes elementos básicos:

a) MySQL es la base de datos **Open Source más popular**, es desarrollada, distribuída y soportada por MySQL AB. Esto significa que cualquiera puede usar y modificar el software, y quien sea pude descargar el software MySQL de Internet y utilizarlo sin pagar nada. Incluso hasta se puede estudiar el código, adaptarlo a determinadas necesidades particulares y volverlo a compilar. Ahora bien, si no se está de acuerdo con los términos de la licencia GPL, se puede comprar una versión con licencia comercial.
b) La página web de mysql http://www.mysql.com provee las **últimas versiones del software** de mysql, sean dbEngines para diversos sistemas operativos, conectores para Microsoft ODBC, JDBC, DBI, interfases gráficas para hacer consultas, manuales en línea, y todo el soporte necesario
c) MySQL es un sistema de administración de base de datos que maneja **grandes cantidades de datos**.
d) MySQL es un sistema de administración de base de datos **relacionales**. Se manejan los datos en tablas separadas en lugar de almacenar todos los datos en un archivo grande, lo que añade velocidad y flexibilidad. También utiliza el lenguaje ANSI/ISO SQL estándar.
e) El servidor MySQL es **rápido, confiable, y fácil de utilizar**. Además esta tecnología se encuentra en constante desarrollo para seguir ofreciendo un rico y útil conjunto de funciones. Su conectividad, velocidad y seguridad hacen del servidor MySQL diseñado para desarrollar bases de datos en Internet.
f) El servidor MySQL puede manejar conexiones en el kernel de forma concurrente (es multithreaded). Esto significa que **puede utilizar múltiples CPU** si están disponibles.
g) MySQL tiene versiones para diferentes plataformas. **Es portable**.
h) MySQL puede realizar operaciones **transaccionales y no transaccionales**.
i) MySQL provee APIs para clientes de **diversos lenguajes de programación**: C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
j) Sistema de **direccionamiento de memoria** rápido, basado en threads.
k) Consultas tipo join de **alta velocidad** utilizando el multi-join de una barrida optimizado.
l) Uso de tablas temporales en memoria.
m) Biblioteca de funciones **SQL de alta eficiencia**.
n) **Código de MySQL probado** con Purify (es un detector comercial de fugas de memoria) así como Valgrind, una herramienta GPL (http://developer.kde.org/~sewardj/)[5]

2.3 Arquitecturas de Sistemas Distribuidos

El problema a resolver planteado en el capítulo 1está situado en un escenario de objetos, o servicios o entidades distribuidas. Por esta razón es conveniente hablar acerca del cómputo distribuido y bien pudiéramos comenzar con una breve historia.

En sus inicios, los primeros programadores codificaban sus algoritmos en lenguaje máquina, y en el mejor de los casos, en lenguaje ensamblador, el cual, a través de un compilador, generaba el código de máquina. El problema que inicialmente se detectó es que los programas en ensamblador no son portables de una tecnología a otra. En ésos días históricos, los programas de computadora en lenguaje ensamblador eran ejecutados en un espacio de memoria sencillo. Los servicios de software en este caso eran subrutinas escritas en lenguaje ensamblador, y estaban comunicadas digitalmente a través de registros de máquina. Los programas escritos de esta forma no eran portables, difíciles de codificar y de mantenimiento costoso.[6]

Es decir, el ensamblador para un fabricante de procesadores, por ejemplo, Motorola, o Texas Instruments, era incompatible con el ensamblador de otro fabricante de procesadores como Zilog o Intel. Por tanto, se tuvo que pensar en otra solución,

[5] WIDENIUS, Michael Monty. et. al., MySQL Reference Manual. pp. 3-6
[6] GLASS, Graham. Web Services: Building Blocks for Distributed Systems, p. 3

como los compiladores de lenguaje de alto nivel. Los compiladores permitieron cierta portabilidad de los programas de una arquitectura a otra

Después que el lenguaje ensamblador se tornó en lenguajes de procedimiento como FORTRAN y COBOL, los cuales proveen una capa de abstracción, los programas pudieron ser escritos de forma más sencilla y corrían en diferentes máquinas. Los servicios de software en este caso eran funciones las cuales eran orquestadas por estructuras de control, y tal como las subrutinas en ensamblador, eran ejecutadas en un sencillo espacio de memoria.[7]

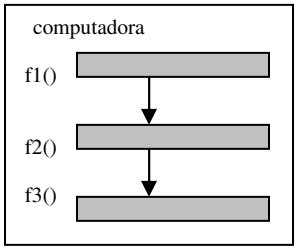


Ilustración 3 Funciones comunicándose dentro de un espacio de memoria simple

Con esto se generó la necesidad de tener, para un mismo lenguaje, por ejemplo, ANSI C, un compilador para cada uno de los procesadores y sistemas operativos en el mercado. Para el ejemplo citado, se llegó a tener un compilador ANSI C para VMS de Digital Micro VAX, para HP UNIX, para Intel con Linux, para Intel con Windows, etc.

Al fenómeno de la multiplicidad de procesadores, sistemas operativos y compiladores se agregó una variable trascendente: los sistemas de cómputo llegaron a tener la capacidad de compartir datos y recursos con la invención de las diversas arquitecturas de red. En este sentido, la industria de la ingeniería de programación también reaccionó con conceptos como Remote Procedure Call (RPC), para que los sistemas de cómputo también pudiesen compartir servicios de software.

La computación en red permitió que los sistemas intercambiaran información en tiempo real en lugar de tener que transferir los datos utilizando cintas magnéticas. El sistema operativo UNIX tenía un protocolo de soporte de red interno Transmisión Control Protocol / Internet Protocol (TCP/IP) y C llegó a ser el lenguaje de programación popular para escribir aplicaciones de red. Una técnica llamada Remote Procedure Call fue inventada para permitir que las funciones escritas en C, FORTRAN o algún otro lenguaje procedural se invoquen mutuamente a través de la red, permitiendo que los servicios de software se comunicaran libremente desde máquinas individuales y colaborar en gran escala. Un protocolo llamado eXternal Data Representation (XDR) permitió que las estructuras de datos complejos fueran enviadas entre las funciones, sin considerar la compatibilidad de plataforma o lenguaje.[8]

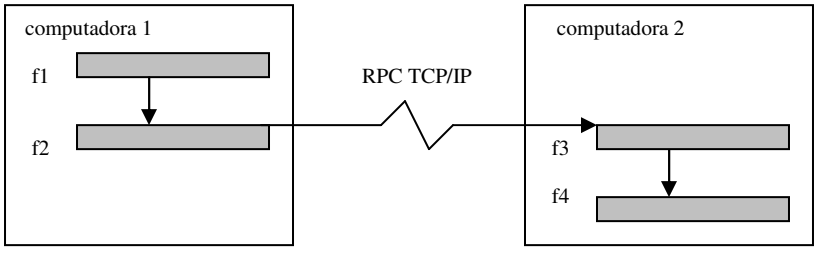


Ilustración 4 Funciones comunicándose entre diferentes espacios de memoria usando RPC

[7] Ibid. p. 3
[8] Ibid. pp. 3,4

El RPC primitivo, sin embargo, no solucionó el problema de la multiplicidad de plataformas existentes en ese tiempo. Uno de los problemas mas graves a los que se enfrentó es un elevado número de plataformas incompatibles entre sí. De hecho, el número de plataformas existentes es proporcional al grado de dificultad para desarrollar una aplicación en un entorno heterogéneo. Es decir, el problema que se llega a encontrar al momento de conectar y presentar una aplicación para usarse en una nueva plataforma en red puede ser el que se tengan dos o más versiones de la misma aplicación. Si por alguna causa se realizan cambios en alguna de las versiones de la aplicación, es obligatorio regresar y modificar todas las versiones y entonces probarlas individualmente y en varias combinaciones para asegurarse que todas ellas funcionan apropiadamente. El grado de dificultad presentado por esta situación se incrementa dramáticamente si el número de diferentes plataformas en la red se incrementa.[9]

Vale la pena detenerse y preguntar: ¿*porqué los sistemas son heterogéneos*? Una razón obvia es que la tecnología avanza con el tiempo. Debido a que las redes tienden a desarrollarse en lugar de ser construidas en un instante, las mejores tecnologías de diferentes periodos de tempo coexisten en la red. En este contexto,"lo mejor" podría referirse a las cualidades como el bajo costo, el alto desempeño, el almacenamiento de datos menos costoso, el mayor número de transacciones por minuto, la más estricta seguridad, los más veloces gráficos, u otras características consideradas importantes al tiempo de compra.

Otra razón es que "en un tamaño no cabe todo", es decir, un tipo de procesador, sistema operativo y compilador no siempre es apropiado para diferentes tipos de tareas. Se requiere combinar una serie de computadoras, sistemas operativos y plataformas para realizar un conjunto de actividades de cómputo dentro de la red.[10]

¿Porqué alguien podría desear desarrollar una aplicación distribuida en primera instancia? La distribución introduce un nuevo conjunto de asuntos dificultosos. Sin embargo, muchas veces no hay elección: algunas aplicaciones por su pura naturaleza son distribuidas a través de múltiples computadoras debido a una o más de las siguientes razones:

- **Los datos son distribuidos.** Algunas aplicaciones deben ejecutarse en múltiples computadoras debido a que los datos que la aplicación debe manejar existe en múltiples computadoras por razones administrativas y de pertenencia física. El dueño podría permitir que los datos sean obtenidos remotamente, pero no almacenados localmente. O quizás los datos no pueden ser situados en la misma computadora, y deben existir en sistemas heterogéneos múltiples por razones históricas.
- **La computación es distribuida.** Algunas aplicaciones se ejecutan en múltiples computadoras a fin de aprovechar el cómputo de múltiples procesadores en paralelo para resolver algún problema[11]. Otras aplicaciones podrían ejecutarse en múltiples computadoras a fin de aprovechar de una característica en particular del sistema. Las aplicaciones distribuidas pueden aprovechar la escalabilidad y heterogeneidad de los sistemas distribuidos.
- **Los usuarios son distribuidos.** Algunas aplicaciones se ejecutan en múltiples computadoras porque los usuarios de la aplicación se comunican entre sí a través de la aplicación. Cada usuario ejecuta una pieza de la aplicación distribuida en su computadora, y los objetos compartidos, típicamente se ejecutan en uno o más servidores.[12]

En relación a lo anterior, los sistemas distribuidos están basados en componentes de diferentes vendedores. La programación basada en componentes ha llegado a ser más popular que nunca. Difícilmente una aplicación es construida en nuestros días de forma que no implique apoyarse en componentes de alguna forma, usualmente de diferentes vendedores. En la medida que las aplicaciones han crecido en sofisticación, la necesidad de apoyarse en componentes distribuidos en computadoras remotas ha crecido también.[13]

Otro obstáculo más que se puede encontrar en un entorno heterogéneo es el hecho de que la heterogeneidad de los sistemas se gesta por la variedad de componentes empaquetados de software conocidos como libraries o bibliotecas, los cuales son reutilizables por diversos lenguajes de programación en diversas plataformas. Al día de hoy escribir una aplicación

[9] VINOSKY, Steve. Advanced CORBA Programming With C++, p.10
[10] Ibid., pp. 9,10
[11] Los problemas de procesamiento pesado son: render gráfico, cálculos astronómicos, cálculo de nóminas, cálculo de contabilidad, cálculos para instituciones financieras, crediticias, de bolsa, etc.
[12] MAGELANG. Introduction to CORBA p.2
[13] SHORT, Scott. Building XML Web Services For The Microsoft .NET Platform p. 1

distribuida robusta de inicio a fin, por ejemplo, una interfase de usuario gráfica personalizada así como los transportes y protocolos de red, es tremendamente difícil para casi cualquier aplicación del mundo real debido a la abrumante complejidad y el número de detalles envueltos en la tarea. Como resultado, los desarrolladores de aplicaciones distribuidas tienden a hacer un uso intensivo de herramientas y "libraries". Esto significa que las aplicaciones distribuidas son por si mismas heterogéneas, la mayoría de las veces fusionadas en un número de aplicaciones y bibliotecas estratificadas. Desafortunadamente, al momento que el sistema crece, se encuentra con la limitante de que las aplicaciones y bibliotecas que las componen deben trabajar juntas, lo que impide fraccionar la aplicación.[14]

Todos estos factores nos revelan que *la heterogeneidad de las redes de computadoras es una realidad con nuevas problemáticas asociadas*. Las redes de computadoras típicamente son heterogéneas. Por ejemplo, la red interna de una compañía pequeña de software podría estar hecha de múltiples plataformas de cómputo. Podría existir un mainframe para manejar el acceso a una base de datos transaccional para ingreso de ordenes, estaciones de trabajo UNIX para proveer entornos de simulación de hardware y un segmento de desarrollo de software, esto es, computadoras personales que corran alguna versión reciente de Windows y provean herramientas de automatización para escritorio de oficina, y otros sistemas especializados como computadoras para control de red, sistemas de telefonía, ruteadores, y equipos de medición. Solo muy pequeñas secciones de una red dada podrían ser homogéneas, pero el grueso de una red es la mayoría de las veces variado y tiende a ser diverso en composición.[15]

¿De qué manera se pueden resumir todas estas necesidades del cómputo de nuestros tiempos? Se pueden citar tres puntos:

- Los factores que conducen a las redes de computadoras heterogéneas son inevitables, por tanto, los desarrolladores prácticos de los sistemas distribuidos, les guste o no, deben hacer frente a la heterogeneidad.
- El software distribuido debe tratar con las fallas de algunos sistemas en la red, fragmentación de la red, problemas asociados con la administración y asignación de los recursos, y riesgos relacionados con la seguridad.
- Con la heterogeneidad de los sistemas, algunos problemas se vuelven mas acotados, y nuevos problemas salen a flote.[16]

Esta realidad inmutable nos lleva a pensar en un conjunto de reglas para atacar la problemática de desarrollo en un sistema heterogéneo, es decir, crear un sistema distribuido. El problema de desarrollar aplicaciones para sistemas distribuidos heterogéneos se resuelve siguiendo dos reglas básicas:

1. **Buscar modelos independientes de plataforma y abstracciones que pueden aplicarse para ayudar una amplia variedad de problemas**. Al usar los modelos y abstracciones apropiadas, se puede proveer una nueva capa de desarrollo de aplicaciones homogénea por encima de toda la complejidad heterogénea distribuida.
2. **Ocultar tanta complejidad de bajo nivel como sea posible, sin sacrificar mucho el rendimiento**. Al ocultar la complejidad de bajo nivel, se permitirá que los desarrolladores de aplicaciones resuelvan sus problemas inmediatos sin tener que pensar en los detalles de bajo nivel de la red para todas las plataformas de cómputo diversas usadas por sus aplicaciones.[17]

Una tercera regla que se añade al margen de las primeras es que los sistemas de objetos distribuidos sean obligatoriamente definidos a partir de la terminología de la programación orientada a objetos, porque en los sistemas de objetos distribuidos todas las entidades son modeladas como objetos. Los sistemas de objetos distribuidos son el paradigma popular para aplicaciones distribuidas orientadas a objetos. Desde que la aplicación se modela como un conjunto de objetos que cooperan entre sí, esta tiene una correspondencia natural a los servicios del sistema distribuido.[18]

En éste trabajo de tesis se toman solamente tres de esos modelos del amplio conjunto de arquitecturas orientadas a la programación de aplicaciones distribuidas: CORBA, RMI y Web Services. En la **sección 2.4** se tocarán con más detalle haciendo una comparación entre esas tres arquitecturas para finalmente seleccionar una de ellas como instrumento

[14] VINOSKY. Op. Cit. p.10
[15] Ibid. p. 9
[16] Ibid. p. 10
[17] Ibid. pp. 10, 11
[18] MAGELANG. Op. Cit., p.3

metodológico a fin de desarrollar la solución al problema inicial: desarrollar un sistema de recepción de reportes de ventas y órdenes de reparación de la industria automotriz.

El hecho de que el cómputo distribuido esté de moda, ¿significa que todas las aplicaciones que se hagan de hoy en adelante se harán en un entorno como éste? No necesariamente. ¿Por qué? Es debido a que se debe analizar si un sistema es candidato a ser diseñado de manera distribuida, ya que el desarrollarlo de esta manera conlleva al menos 4 preocupaciones adicionales:

1. La comunicación entre objetos en el mismo proceso **es más rápida** que la comunicación entre objetos de diferentes máquinas. Esto implica que se debe evitar diseñar aplicaciones distribuidas en las cuales dos o más objetos distribuidos tengan interacciones muy cercanas. Esto sucede, dichos objetos deben pertenecer al mismo proceso.
2. Si dos objetos están distribuidos a través de barreras de proceso, los objetos pueden **fallar independientemente**. En este caso, el diseñador de los objetos debe saber la conducta de cada objeto, en el caso de que el otro objeto falle.
3. Cada objeto distribuido opera en un diferente **hilo de control**. Un objeto distribuido puede tener múltiples clientes concurrentes. Como desarrollador del objeto y el desarrollador de los clientes, se debe considerar el acceso concurrente a los objetos y usar los mecanismos de sincronía necesarios.
4. Cuando los objetos están en diferentes computadoras, se deben usar mecanismos de **seguridad** para validar la identidad del otro objeto.[19]

Como se puede ver, los sistemas distribuidos no es la arquitectura ideal para todas las aplicaciones.

Se puede decir que es menos comprometedor desarrollar sistemas monolíticos o no distribuidos (en donde la interfaz de usuario, base de datos y programación de reglas de negocios se encuentran en el mismo entorno ó proceso) contra desarrollar sistemas distribuidos (arquitectura cliente robusto/servidor ligero; cliente ligero/servidor robusto; n capas). Para explicar esto con un grado más de detalle podemos marcar las diferencias entre los sistemas distribuidos y los monolíticos, ya que los desarrolladores de aplicaciones distribuidas deben tratar con un número de asuntos que no tienen lugar en programas locales (en donde toda la lógica se ejecuta en el mismo proceso). La siguiente tabla resume algunas de las diferencias básicas entre los objetos que están localizados en el mismo proceso, y objetos que interactúan con otro proceso o máquina.[20]

Rasgo	**Mismo proceso**	**Procesos distribuidos**
Comunicación	Rápida	Lenta
Fallas	Los objetos fallan conjuntamente	Los objetos fallan de manera separada. La red puede partirse
Acceso concurrente	Solo con múltiples threads o hilos	Sí por naturaleza de arquitectura.
Seguridad	Si	Deben definirse diferentes capas de seguridad

Tabla 1 Comparación entre procesos distribuidos y no distribuidos

Para el caso que compete a esta tesis, es obligatorio utilizar alguna de las arquitecturas orientadas a la programación de aplicaciones distribuidas, debido a las siguientes razones:

1. La industria automotriz cuenta con su infraestructura de red privada (sea esta propia o provista por algún tercero), cuyo ancho de banda está repartido en datos, voz y video, lo que llega a generar una velocidad de transferencia de información variable, entre el corporativo y los concesionarios;
2. La industria automotriz funciona con el modelo de mercadotecnia corporativo-franquicia, lo que obliga a las franquicias a reportar regularmente datos críticos, como el volumen de ventas –quizás diario, muy probablemente semanal y en el peor de los casos mensual- así como mantener informado al corporativo acerca de las órdenes de reparación en días hábiles.
3. Las franquicias se encuentran en diferentes puntos geográficos de la República Mexicana, o en el Extranjero.
4. Las franquicias no tienen la misma infraestructura de red de área local.

[19] Ibid. p. 3
[20] Ibid. pp. 2,3

5. Las franquicias están soportadas en plataformas diversas –variando el procesador, tipo de sistema operativo o versión de sistema operativo, por mencionar solo unos ejemplos-
6. Las franquicias son administradas con productos de software de diferentes proveedores certificados, productos que se adaptan a la plataforma, infraestructura y presupuesto del distribuidor automotriz, y que no siempre están codificados en el mismo lenguaje de programación de franquicias primas.
7. En relación con lo anterior, se desean ligar eventos de dichos productos de software con el envío de la información, para evitar la duplicidad de trabajo (es decir, capturar la información de ventas y órdenes en el sistema local y después tener que volverla a capturar en el sistema corporativo).
8. Se debe tener una forma homogénea para enviar las órdenes de reparación y los reportes de ventas, independientes de plataforma, lugar geográfico, lenguaje de programación, etc.
9. Se desea que el corporativo reciba la información crítica lo más rápido posible, sin fallas o errores, concurrentemente y de forma segura.

Se puede concluir entonces que uno de los componentes que integran la solución del problema a resolver será una de las arquitecturas de objetos distribuidos. ¿Cuáles son éstas? ¿Qué características tienen? Al seleccionar una de ellas¿Cómo saber que fue la mejor? La siguiente sección contestará dichas preguntas.

2.4 Alternativas en arquitecturas de sistemas distribuidos

Si bien el seleccionar una base de datos o un lenguaje de programación no es difícil, si lo es el seleccionar una arquitectura de objetos distribuidos para desarrollar la aplicación. La siguiente sección está dedicada a hacer un análisis comparativo de las arquitecturas comúnmente utilizadas para desarrollar aplicaciones en entornos distribuidos, y dará las razones por las que fue elegido el modelo de XML Web services.

2.4.1 Protocolos estándar: CORBA

CORBA, o Common Object Request Broker Architecture, es una arquitectura estándar para sistemas de objetos distribuidos. Esta permite que operen entre sí un conjunto de objetos heterogéneos y distribuidos.[21]

CORBA provee interfaces y modelos de programación de plataforma independiente para aplicaciones de cómputo distribuidas, orientadas a objetos. Su independencia de lenguajes de programación, plataformas de cómputo y protocolos de red hacen a CORBA idóneo para el desarrollo de nuevas aplicaciones y su integración en los sistemas distribuidos existentes.[22]

Una de las metas de la especificación CORBA es que los clientes y las implementaciones a objeto sean portables. La especificación CORBA define una Application Programmer's Interface (API) para los clientes de objetos distribuidos, así como una API para la implementación de un objeto CORBA. Esto significa que el código escrito por un vendedor de productos CORBA puede, con un mínimo esfuerzo, ser reescrito para trabajar con un vendedor de producto diferente. Sin embargo, la realidad de los productos CORBA en el mercado ahora es que los clientes CORBA son portables, pero las implementaciones a objeto necesitan algo de retrabajo para trasladarse de un producto CORBA a otro.[23]

¿Porqué es CORBA una buena opción para diseñar y escribir sistemas?

- CORBA está al día de hoy bien establecida en el camino central del desarrollo de software y ha encontrado una fenomenal aceptación de la industria.
- CORBA está soportada en casi toda combinación de hardware y sistemas operativos en existencia.
- CORBA esta disponible desde un amplio número de vendedores, aún como freeware. Soporta un amplio número de lenguajes de programación y está siendo utilizada al día de hoy para crear aplicaciones de misión crítica en industrias de diverso giro como cuidado de la salud, telecomunicaciones, bancos y manufactura.

[21] Ibid. p.3
[22] VINOSKY. Op. Cit., p. 14
[23] MAGELANG. Op. Cit., p. 4

- El incremento de la popularidad de CORBA ha creado un incremento correspondiente en la demanda de ingenieros de software que sean competentes en la tecnología.[24]

¿Qué contiene la especificación de CORBA? La especificación de CORBA, escrita y mantenida por OMG (consorcio de más de 800 empresas), provee un conjunto balanceado de abstracciones flexibles, y servicios concretos necesarios para construir soluciones prácticas para los problemas asociados con la computación heterogénea distribuida.[25]

La especificación central del OMG es el concepto de Object Management Architecture (OMA), quien provee un marco de trabajo de arquitectura completa que es tanto lo suficientemente rico como flexible para acomodar una amplia variedad de sistemas distribuidos. OMA usa dos modelos relacionados para describir como los objetos distribuidos y sus interacciones entre ellos pueden ser especificadas en caminos de plataforma independiente.

- El Modelo de Objetos define como las interfaces de objetos distribuidos a través de un ambiente heterogéneo son descritos y
- El Modelo de Referencias caracteriza las interacciones entre los objetos del ambiente heterogéneo. [26]

Otro gran logro de éste modelo de OMG es la definición de IDL. Los mapeos de lenguaje especifican como IDL es traducido en diferentes lenguajes de programación. Para cada construcción en IDL, un mapeo de lenguaje define cuales recursos para el lenguaje de programación son utilizados para hacer la construcción disponible para las aplicaciones. Por ejemplo, en C++, las interfaces IDL son mapeadas en clases, y las operaciones son mapeadas en funciones que son miembros de esas clases. Similarmente, en Java, las interfases IDL son mapeadas a interfaces Java públicas.

Las referencias a objetos en C++ mapean a constructores que soportan la función `operator->` (esto es, ya sea un apuntador a clase o un objeto de una clase con la función miembro `operador->` con sobrecarga). Las referencias a objetos en C, por otra parte, mapean a apuntadores opacos (del tipo `void *`) y las operaciones son mapeadas en funciones C cada que se requiere una referencia a objeto opaca y se toma el primer parámetro.

Los mapeos de lenguaje también especifican cómo las aplicaciones usan los recursos del ORB y como las aplicaciones en el servidor implementan los sirvientes.[27]

¿Qué lenguajes de programación ya consideran a CORBA IDL? Los mapeos de lenguaje OMG IDL existen para varios lenguajes de programación. OMG ha estandarizado mapeos de lenguaje para C, C++, Smalltalk, COBOL, Ada y Java. Otros mapeos de lenguaje también existen, -por ejemplo, también se han definido mapeos de forma independiente para Eiffel, Modula 3, Perl, Tcl, Objective-C y Python- pero todavía no están bien estandarizados por OMG.[28]

¿Por qué son tan necesarios los mapeos de IDL a los lenguajes de programación? Los mapeos de lenguaje IDL son críticos para el desarrollo de aplicaciones. Estos proveen realizaciones concretas de los conceptos abstractos y los modelos proporcionados por CORBA. Un mapeo de lenguaje completo e intuitivo hace mas directo desarrollar aplicaciones CORBA en ese lenguaje; por otra parte, un mapeo de lenguaje pobre, incompleto o ineficiente bloquea seriamente el desarrollo de la aplicación CORBA. Las especificaciones oficiales OMG del mapeo de lenguaje, por tanto, sufren revisiones periódicas para hacer mejoras que aseguren su efectividad.[29]

Esta característica de CORBA hace que los sistemas distribuidos sean independientes del lenguaje de programación. La existencia de mapeos de lenguaje OMG IDL múltiples significa que los desarrolladores pueden implementar diferentes porciones de sistemas distribuidos en diferentes lenguajes. Por ejemplo, un desarrollador podría escribir una aplicación en C++ para servidor de alto rendimiento para eficiencia, y escribir clientes en Applets de Java, por lo que estos podrían

[24] VINOSKY. Op. Cit., p. 1
[25] Ibid., p. 11
[26] Ibid.
[27] Ibid., p. 19
[28] Ibid., pp. 19, 20
[29] Ibid., p. 20

descargarse vía Web. La independencia de lenguaje de CORBA es la llave valiosa que permite la integración de la tecnología para sistemas heterogéneos.[30]

Ahora bien, ¿cómo se concibe un objeto desde el punto de vista del Modelo de Objetos en CORBA? El Modelo de Objetos define un objeto como una entidad encapsulada con una identidad distinta inmutable cuyos servicios son utilizados solamente a través de interfases bien definidas. Los clientes utilizan los servicios de objetos al enviar requisiciones (requests) al objeto. Los detalles de implementación del objeto y su localización permanecen escondidos para los clientes.[31]

Por otra parte, ¿en qué consiste el modelo de referencias de CORBA? Este modelo provee categorías de interfases que son agrupaciones generales de interfases de objetos. Todas las categorías de interfases están conceptualmente ligadas por un Object Request Broker (ORB). Generalmente, un ORB habilita la comunicación entre clientes y objetos, activando de forma transparente esos objetos que no están corriendo cuando las requisiciones son enviadas a ellos. El ORB también provee una interfaz que puede ser utilizada directamente por clientes y también por objetos.[32]

¿Qué características tienen las referencias a Objeto de CORBA? Dichas son análogas a apuntadores de instancia de clases C++, pero pueden denotar objetos implementados en procesos diferentes (posiblemente en otras máquinas) así como objetos implementados en el propio espacio de direcciones del cliente. Excepto por esta capacidad de direccionamiento distribuido, las referencias a objeto tienen semánticas muy similares a las que tienen los apuntadores de instancia de clases C++.

- Cada referencia a objeto identifica exactamente una instancia de objeto.
- Varias referencias diferentes pueden denotar el mismo objeto.
- Las referencias pueden ser nil (apuntar a ningún lado).
- Las referencias pueden quedar colgadas (como en C++, los apuntadores que apuntan a instancias borradas)
- Las referencias son opacas (el cliente no está autorizado para ver dentro de su contenido).
- Los tipos de datos de las referencias están fuertemente definidos.
- Las referencias soportan utilización tardía.
- Las referencias pueden ser persistentes.
- Las referencias pueden ser interoperables.[33]

¿Existe en CORBA un protocolo de estandarización de referencias? La interoperabilidad de ORB –que se analizará más adelante su definición- requiere formatos estandarizados de referencia de objetos. Las referencias a objetos son opacas para las aplicaciones, pero estas contienen información que los ORBs necesitan a fin de establecer las comunicaciones entre clientes y objetos destino. El formato estándar de referencia de objetos, llamado Interoperable Object Reference (IOR), es lo suficientemente flexible para almacenar información de casi cualquier interORB protocol imaginable. Un IOR identifica uno o más protocolos soportados y, para cada protocolo, contener información específica a ese protocolo. Este arreglo permite que nuevos protocolos sean agregados a CORBA sin romper las aplicaciones existentes. Para IIOP, un IOR contiene un nombre de host, un número de puerto de TCP/IP, y un object key que identifica el objeto destino en la combinación dada de nombre de host y puerto.[34]

En este punto y al haber mencionado los elementos implícitos en la arquitectura CORBA, podríamos trazar un esquema muy general del funcionamiento de esta arquitectura para objetos distribuidos. El paradigma básico de CORBA es aquel que, en una petición de servicios, se formula a un objeto distribuido. Todo lo demás definido por la OMG está en términos de este paradigma básico.

Los servicios que un objeto provee están dados por su interfaz. Las interfases son definidas por el Interfaz Definition Language (IDL) de OMG. Los objetos distribuidos son identificados por referencias a objetos, los cuales son descritos y clasificados por las interfases de IDL. Se tiene el siguiente ejemplo:

[30] Ibid.
[31] Ibid., p. 11
[32] Ibid., pp. 11, 12
[33] Ibid., p. 25
[34] Ibid., p. 23

La figura muestra una petición. Un cliente retiene la referencia a objeto para un objeto distribuido. La referencia a objeto está descrita por su interfaz. En la figura la referencia a objeto está descrita por la interfaz Rabbit. El Object Request Broker, u ORB envía la petición al objeto y regresa algún resultado al cliente.

Aquí, la petición jump regresa una referencia a objeto especificada por la interfaz AnotherObject.[35]

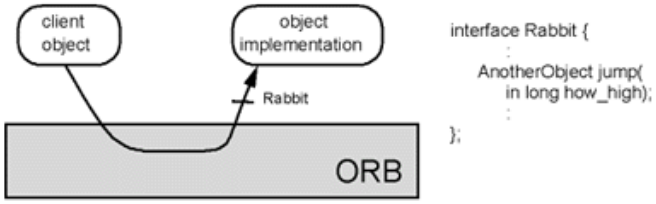


Ilustración 5 Arquitectura CORBA simplificada

A pesar de la utilidad del diagrama anterior para trazar un marco conceptual, se requiere otro diagrama con una descripción formal de la arquitectura CORBA. El diagrama es el siguiente:

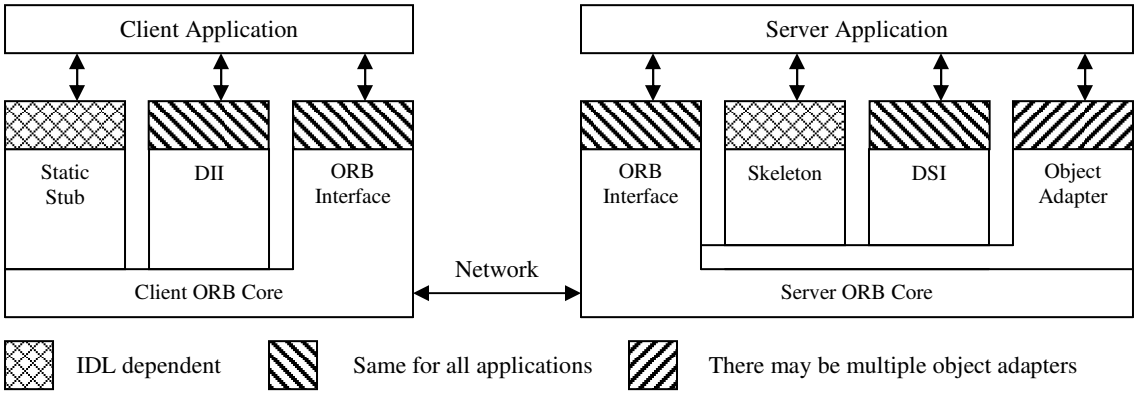


Ilustración 6 Descripción formal de la arquitectura CORBA

Este diagrama muestra **cómo las requisiciones a objetos distribuidos fluyen** de la siguiente manera:

1. Para hacer la requisición, el cliente puede utilizar o bien static stubs compilados en C++ a partir de la definición de interfase de objeto, o usando el DII, Dinamic Invocation Interface. De ambas formas, el cliente dirige la requisición en el núcleo ORB ligado dentro de su proceso.
2. El núcleo ORB del cliente transmite la requisición al núcleo ORB ligado con la aplicación del servidor.
3. El núcleo ORB del servidor reparte la requisición al Object Adapter que ha creado dicho objeto.destino.
4. El Object Adapter a su vez, reparte la requisición al sirviente que está implementando el objeto destino. Así como el cliente, el servidor puede seleccionar entre mecanismos de atención estáticos y dinámicos para sus sirvientes. Podría darse el caso en que descanse sobre static skeletons compilados en C++ de la definición de interfase de objeto, o sus sirvientes pueden usar el DSI (Dynamic Skeleton Interface)
5. Después que el sirviente lleva a cabo la requisición, regresa la respuesta a la aplicación del cliente.[36]

[35] MAGELANG. Op. Cit., p.4
[36] VINOSKY. Op. Cit., pp. 16,17

En la figura anterior aparecen varios elementos, que a pesar que son definidos en la programación orientada a objetos, son redefinidos por la arquitectura CORBA. Dicha **terminología única asociada con CORBA es la clave para un firme entendimiento de la arquitectura**. Los elementos son los siguientes:

1. **CORBA Object**. Es una entidad virtual capaz de ser localizable por un ORB y teniendo requisiciones del cliente invocados en ella. Es virtual en el sentido de que no existe realmente al menos que sea creada concretamente por una implementación escrita en algún lenguaje de programación. La construcción de un objeto CORBA por un lenguaje de programación es análogo a la forma en que la memoria virtual de una PC no existe -en un sistema operativo, pero es simulada usando memoria física.
2. **CORBA Target Object**. Dentro del contexto de la invocación del CORBA request, es el objeto CORBA que es el objetivo de dicha requisición. El modelo de objetos CORBA es un modelo de atención sencilla en el cual el objeto objetivo para una requisición está determinado solamente por la referencia del objeto usada para invocar la requisición.
3. **CORBA Client**. Es una entidad que invoca la requisición de un objeto CORBA. Un cliente podría existir en un espacio de direcciones que se encuentra completamente separado del objeto CORBA, o el cliente y el objeto CORBA podrían existir dentro de la misma aplicación. El término cliente tiene significado solamente dentro del contexto de una requisición en particular debido a que la aplicación que es el cliente de una requisición podría ser el servidor para otra requisición.
4. **CORBA Server**. Es una aplicación en la cual uno o más objetos CORBA existen. Así como en los clientes, este término es significativo solamente en el contexto de una requisición en particular.
5. **CORBA Request**. Es la invocación de una operación en un objeto CORBA por un cliente. Las requisiciones fluyen del cliente al objeto objetivo en el servidor, y el objeto objetivo envía los resultados de regreso en respuesta si la requisición requiere una.
6. **CORBA Object Reference**. Es un manejador utilizado para identificar, localizar, y dar dirección a un objeto CORBA. Para los clientes, las referencias a objeto son entidades opacas. Los clientes usan referencias a objetos para direccional requisiciones a objetos, pero estos no pueden crear referencias a objetos de sus partes constituyentes, ni tampoco pueden abrir o modificar el contenido de una referencia a objeto. Una referencia a objeto se refiere solamente a un objeto CORBA sencillo.
7. **CORBA servant**. Un sirviente es una entidad en un lenguaje de programación determinado que implementa uno o más objetos CORBA. Se dice que los sirvientes encarnan los objetos CORBA debido a que ellos proveen cuerpos, o implementaciones, para dichos objetos. Los sirvientes existen dentro del contexto de un servidor de aplicaciones. En C++, los sirvientes son instancias de objetos de una clase en particular.[37]

El punto 5 del bloque anterior habla acerca de requests, o peticiones, para disparar operaciones en objetos remotos. Pues bien, este elemento tiene un ciclo de ejecución. En este proceso los clientes manipulan los objetos al enviarles mensajes. El ORB envía un mensaje a un objeto siempre y cuando el cliente invoque una operación. La referencia a objeto actúa como un manejador que identifica de forma única el objeto destino y encapsula toda la información requerida por el ORB para enviar el mensaje al destino correcto.

Cuando un cliente invoca una operación a través de una referencia de objeto, *el ORB realiza lo siguiente*:

- Localizar el objeto destino.
- Activar la aplicación en el servidor si el servidor no se encuentra corriendo actualmente.
- Transmitir algunos argumentos para la llamada al objeto.
- Activar un sirviente para el objeto si es necesario.
- Esperar a que la petición se complete.
- Regresar algunos parámetros “out” e “inout” y el valor de regreso al cliente cuando la llamada se complete de forma satisfactoria.
- Regresar una excepción (incluyendo algún dato contenido en la excepción) al cliente cuando la llamada falle.[38]

Existen 4 tipos de requisiciones y son:

[37] Ibid., p.14
[38] Ibid., p. 23

- Cuando un cliente invoca una requisición **síncrona**, se detiene mientras espera una respuesta. Este tipo de requisiciones son idénticas a las llamadas a procedimientos remotos (RPC).
- Un cliente que invoca una requisición **síncrona diferida** envía la requisición, continúa el procesamiento, y entonces al final solicita la respuesta. Actualmente, este tipo de requisición puede ser utilizado solamente a través del DII, Dynamic Invocation Interface.
- CORBA también provee requisiciones **de una vía**, la cual es la requisición del menor esfuerzo, que puede no ser entregada al objeto destino, y no se le permite tener respuestas. A los ORB se les permite depositar silenciosamente requisiciones de una vía si existe congestión de red o si las fallas en determinados recursos pueden causar que el cliente se detenga mientras la requisición es enviada.
- Una versión futura de CORBA (probablemente la versión 3.0) soportará **requisiciones asíncronas** que pueden ser utilizadas para permitir clientes que se conecten de forma ocasional, así como permitir que algún servidor se conecte con un tercero. Esto también agregará soporte para hacer llamadas síncronas diferidas usando static stubs así como el DII.[39]

Los Requests en CORBA están compuestos por muchas *características ventajosas* que convierte un sistema de objetos distribuidos en un sistema de objetos homogéneos. Por nombrar algunas de ellas se tiene:

- **Transparencia de ubicación**. El cliente no conoce ni le importa si el objeto destino es local a su propio espacio de direcciones, ha sido implementado en un proceso diferente dentro de la misma máquina, o es construido en un proceso en una computadora diferente. Los procesos del servidor no son obligados a permanecer en la misma computadora para siempre. Eventualmente podrán moverse alrededor de la compañía, de máquina a máquina sin que los clientes lleguen a darse cuenta de esto.
- **Transparencia del servidor**. El cliente no necesita saber cual servidor implementa que objetos.
- **Independencia del lenguaje**. El cliente no se preocupa por el lenguaje que ha sido utilizado por el servidor. Por ejemplo un cliente C++ puede llamar un programa en Java sin que se percate de ello. La implementación de lenguaje para objetos puede ser cambiada para objetos existentes, sin afectar en lo mínimo a los clientes.
- **Independencia de programación/construcción.** El cliente no sabe como funciona la implementación. Por ejemplo, el servidor puede implementar sus objetos propiamente con sirvientes C++, o el servidor puede implementar sus objetos usando técnicas no orientadas a objetos (tales como implementar objetos como segmentos de datos). El cliente ve la misma semántica consistente orientada a objetos, sin tomar en consideración de cómo los objetos son implementados en el servidor.
- **Independencia de arquitectura**. El cliente no tiene porqué tomar en consideración la arquitectura del CPU que es utilizada por el servidor y está protegido de detalles como ordenamiento de bytes y conformidad de estructura.
- **Independencia del sistema operativo**. El cliente no está al tanto de que sistema operativo es utilizado por el servidor. El servidor podría aún estar implementado sin el soporte de un sistema operativo –por ejemplo, un programa incrustado en modo real-.
- **Independencia de protocolo**. El cliente no conoce que protocolo de comunicación es utilizado para enviar los mensajes. Si varios protocolos están disponibles para comunicarse con el servidor, el ORB de forma transparente selecciona un protocolo al tiempo de ejecución.
- **Independencia de transporte**. El cliente es ignorante acerca de la capa de transporte y enlace de datos utilizado para transmitir mensajes. Los ORBs pueden de una forma transparente usar varias tecnologías de red tales como Ethernet, ATM, Token Ring, o líneas seriales.[40]

En este punto vale la pena destacar que los CORBA requests no solamente pueden ser clasificados en síncronos, de una vía y asíncronos. **También pueden ser clasificados por la forma en que se construyen, en "static" y "dynamic"**. Hablando sobre los requests "estáticos" se parte del hecho que OMG IDL es traducido en stubs y skeletons de un lenguaje de programación específico, los cuales son a su vez compilados, para que formen parte de las aplicaciones. Compilar stubs y skeletons en una aplicación genera un conocimiento estático de los tipos de datos del lenguaje de programación, y funciones mapeadas de las descripciones de IDL de objetos remotos. Un stub es una función del lado del cliente que permite que llamado de la requisición sea hecho a través de una llamada de función local. En C++, un stub CORBA es una función

[39] Ibid., p. 17
[40] Ibid., pp. 24,25

miembro de una clase. El objeto local C++ que soporta las funciones del stub es frecuentemente llamada PROXY debido a que esta representa el objeto destino remoto de la aplicación local. Similarmente, un skeleton es una función del lado del servidor que permite que el llamado de una requisición sea recibida por el servidor para ser repartida al sirviente apropiado.

Los desarrolladores que escriben aplicaciones en lenguajes estáticamente definidos como C++ usualmente *prefieren usar la aproximación de invocación estática* debido a que provee un modelo de programación más natural.[41]

¿Qué es un CORBA PROXY, detalladamente hablando? Cuando una referencia es recibida por el cliente, el runtime en el lado del cliente genera una instancia del Proxy object (o Proxy, para acortar) en el espacio de direcciones del cliente. Un Proxy es una instancia de C++ que provee al cliente la interfaz al objeto destino. La interfaz en el Proxy es la misma que la interfaz en el objeto remoto; cuando el cliente invoca una operación en el Proxy, el Proxy envía el mensaje correspondiente al sirviente remoto. En otras palabras, el Proxy delega las peticiones al correspondiente sirviente remoto, y se comporta como un embajador local para el objeto remoto, como se muestra en la siguiente figura[42]:

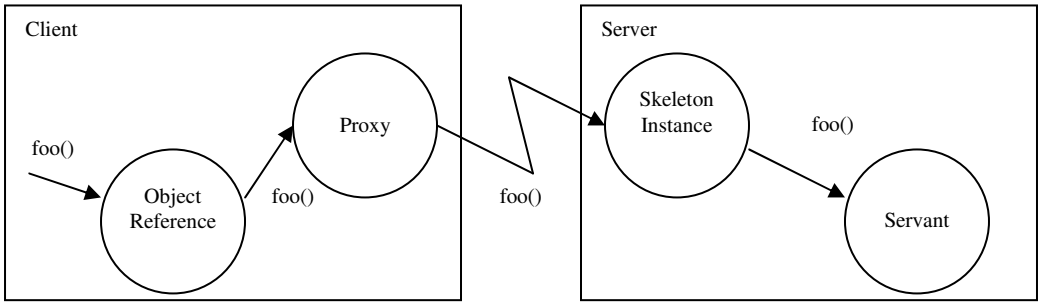


Ilustración 7 El Proxy se comporta como un embajador local para el objeto remoto

¿Qué funciones tiene el PROXY de CORBA? En los escenarios remotos, o en los escenarios donde el cliente y el servidor comparten el mismo espacio de direcciones, el Proxy delega la *invocación de las operaciones hechas por el cliente al servidor*. En el escenario remoto, el Proxy envía la requisición sobre la red, mientras que en el mismo espacio de direcciones, la requisición es atendida utilizando llamados a funciones C++ comunes. La interacción entre el skeleton y el servant en el caso remoto es normalmente implementada como el llamado a una función virtual C++.[43]

¿Cómo se construye un PROXY en CORBA? La instancia del Proxy equipa al cliente con una interfaz que es específica al tipo de objeto que está siendo utilizado. La clase del Proxy es generada de la definición de IDL de la interfaz correspondiente e implementa el stub a través del cual el cliente atiende las llamadas .Esta aproximación asegura la calidad de los tipos de datos: el cliente no puede invocar una operación al menos que contenga un proxy del tipo de datos correcto debido a que solamente ese Proxy tiene la función miembro correcta.[44]

En contraste con los requests estáticos a través de PROXYs, se tienen los **requests dinámicos**. Esta aproximación envuelve la construcción y el envío de requisiciones CORBA en tiempo de ejecución, en lugar de tiempo de compilación (como en la aproximación estática). Debido a que no se tiene la información del tiempo de compilación, la creación e interpretación de las requisiciones en tiempo de ejecución requiere el acceso a servicios que pueden proveer información acerca de las interfases y tipos de datos. La aplicación puede obtener esta información preguntando a un operador humano con una "Guided User Interfase", o una herramienta gráfica con wizards y tutoriales. Alternativamente, se puede obtener por programa del Repositorio de Interfases, un servicio que provee el acceso en tiempo de ejecución a las definiciones IDL.

[41] Ibid., pp. 20,21
[42] Ibid., p. 31
[43] Ibid., p. 32
[44] Ibid.

La aproximación dinámica puede ser muy útil para aplicaciones como gateways y bridges, que deben recibir y reenviar las requisiciones sin necesidad de tener los conocimientos en tiempo de compilación de los tipos de datos e interfases envueltas.[45]

Al desear enviar CORBA requests estáticos o dinámicos a un ORB para invocar una operación en un objeto distribuido, todo cliente debe conocer la interfaz ofrecida por el objeto. Una interfaz de objeto está compuesta de operaciones que el mismo soporta y los tipos de datos que pueden ser transmitidos a y desde esas operaciones. Los clientes también requieren conocer acerca del propósito y la semántica de las operaciones que desean invocar.[46]

¿Cómo son definidas las Object Interfaces en CORBA? En CORBA, las object interfaces son definidas en el OMG Interface Definition Language (IDL). A diferencia de C++ o Java, IDL no es un lenguaje de programación, por lo que los objetos y las aplicaciones no pueden ser generadas en IDL. El único propósito del IDL es permitir que las interfaces de objeto sean definidas de tal forma que sea independiente a algún lenguaje de programación en particular. *Este arreglo permite que las aplicaciones implementadas en diferentes lenguajes de programación puedan interoperar*. La independencia de lenguaje de programación que tiene IDL es crítica en la meta de CORBA para soportar sistemas heterogéneos y la integración de aplicaciones desarrolladas de manera separada.[47]

¿Qué objetivos primarios persigue el IDL? El OMG IDL es el mecanismo fundamental de abstracción para *separar las interfases de objeto de sus implementaciones*. OMG IDL establece un contrato entre el cliente y el servidor que describe los tipos y las interfaces de objeto usadas por una aplicación. Esta descripción es independiente de su lenguaje de implementación, por lo que no importa si el cliente está escrito en el mismo lenguaje que el servidor.[48]

¿Qué especificaciones reúne IDL? OMG IDL soporta tipos simples de datos, tales como enteros con y sin signo, caracteres, boléanos y cadenas, así como tipos de datos construidos como enumeraciones, estructuras y uniones discriminadas, secuencias (vectores de una sola dimensión) y excepciones. Estos tipos son utilizados para definir los tipos de parámetros y los tipos de datos de retorno para operaciones, los cuales están definidos en turno dentro de las interfaces. IDL también provee un módulo de construcción utilizado para propósitos de alcance en el nombre de los objetos.

IDL no puede ser utilizado para escribir detalles de implementación en las aplicaciones. No provee estructuras de control o variables, por lo que no puede ser compilado o interpretado en un programa ejecutable. Está diseñado exclusivamente para declarar interfases para objetos y definir los tipos de datos utilizados para comunicarse con los objetos.[49]

Estas definiciones escritas en el lenguaje IDL pueden ser compiladas para generar subrutinas de tipo stub y skeleton en casi cualquier lenguaje. El compilador traduce las definiciones independientes de lenguaje en definiciones de tipo específicas de cada lenguaje, así como un Application Program Interface (API). Estos tipos y APIS son utilizados por el desarrollador para proveer funcionalidad a la aplicación y para interactuar con el ORB. Los algoritmos de traducción para varios lenguajes de implementación están especificados por CORBA y son conocidos como mapeos de lenguaje. Actualmente, CORBA define mapeos de lenguaje para C, C++, SmallTalk, COBOL, Ada y Java. Esfuerzos independientes están en vías de proveer mapeos de lenguaje adicionales para Eiffel, Modula 3, Perl, TCL, Pitón, Dylan, Oyeron, Visual Basic y Objective-C. Algunos de estos mapeos podrían eventualmente convertirse en estándares.

Debido a que IDL describe interfases pero no implementaciones, es puramente un lenguaje declarativo. No hay forma de escribir sentencias ejecutables en IDL, y no hay forma de decir algo acerca del estado de un objeto (ejecución y estado son asuntos de implementación).[50]

[45] Ibid., pp. 20, 21
[46] Ibid., p. 17
[47] Ibid.
[48] Ibid., p. 51
[49] Ibid., pp. 18, 19
[50] Ibid., p. 52

¿Qué partes tiene IDL? Las definiciones de IDL se enfocan en las interfases de objeto, las operaciones soportadas por esas interfases, y excepciones que pueden ser activadas por las operaciones. Esto requiere por mucho un poco de maquinaria de soporte; en particular, una gran parte de IDL concierne a la definición de los tipos de datos. Esto es debido a que los datos pueden ser intercambiados entre el cliente y el servidor solo si sus tipos están definidos en IDL. No se puede intercambiar datos arbitrarios de C++ entre el cliente y el servidor porque esto destruiría la independencia del lenguaje de CORBA. Sin embargo, siempre se puede crear una definición de tipo de IDL que corresponda a los datos de C++ que se desea enviar, y entonces se puede transmitir el tipo del IDL.[51]

Para cada lenguaje existe un compilador de IDL hacia dicho lenguaje. ¿Qué productos entrega el IDL Compiler? Un compilador IDL produce archivos fuente que deben ser combinados con el código de la aplicación para producir ejecutables del cliente y del servidor. Se presenta una vista conceptual de este proceso debido a que CORBA no estandariza el ambiente de desarrollo. *Esto implica que los detalles, tales como los nombres y el número de archivos fuente generados, varían de un ORB a otro*. Sin embargo, los conceptos son los mismos para todos los ORBs y los lenguajes de implementación.

El resultado del proceso de desarrollo es un ejecutable en el cliente y en el servidor. Estos ejecutables pueden ser instalados en cualquier parte, siempre y cuando estos han sido desarrollados usando el mismo ORB o diferentes ORBs aunque estos hayan sido implementados usando el mismo o diferente lenguaje. La única restricción es que las máquinas que alojarán esas aplicaciones deben proveer el ambiente de ejecución necesario, tal como algunas librerías dinámicas requeridas, y que la conectividad pueda establecerse entre ellas.[52]

Aquí hace falta definir un elemento más y este es el "Object Adapter". ¿Qué función tienen estos? En CORBA, los Object Adapters sirven como el pegamento entre los sirvientes y el ORB. Como se describe en el patrón de diseño de los adaptadores, el cual es independiente de CORBA, un Object Adapter es un objeto que adapta la interfaz de un objeto en una interfaz diferente esperada por el que ha originado la llamada. En otras palabras, un object adapter es un objeto interpuesto que delega, para permitir que el iniciador de la llamada haga ciertas peticiones en un objeto sin conocer la interfaz del objeto verdadera.[53]

¿Qué papel crítico juegan en este escenario los CORBA Object Adapters? Dichos complementan al ORB porque:

1. Crean referencias a objetos, las cuales permiten que los clientes direccionen objetos.
2. Se aseguran que cada objeto destino sea encarnado o implementado por un sirviente.
3. Toman requisiciones enviadas por el ORB del lado del servidor y después los dirigen a los sirvientes que están implementando cada uno de los objetos destino.

Sin los object adapters, el ORB debería proveer directamente estas características, además de otras responsabilidades. Como resultado, podría tenerse una interfaz muy compleja que haría dificultosa la administración para OMG, y el número de estilos de implementación posibles de los sirvientes podría estar limitado.[54]

¿Cómo se implementan las operaciones en el Object Adapter? En C++, por tomar un ejemplo, los sirvientes son instancias de objetos de C++. Son definidos típicamente al derivarse de las clases skeleton producto de compilar las definiciones de interfaz IDL. Para construir las operaciones, se deben sobrescribir funciones virtuales de la clase skeleton base. Entonces se registran esos sirvientes C++ con el object adapter para permitir que este atienda peticiones a los sirvientes, cuando los clientes invocan requisiciones en los objetos incorporados por esos sirvientes.[55]

Ahora bien, ¿Qué funciones y características tiene el ORB dentro de la arquitectura CORBA? El ORB es un servicio distribuido que implementa la petición al objeto remoto. Este localiza el objeto remoto en la red, comunica la petición al objeto, espera el resultado y cuando está disponible, comunica tales resultados de regreso al cliente.

---

[51] Ibid.
[52] Ibid., pp. 52, 53
[53] Ibid., p. 21
[54] Ibid., p. 21
[55] Ibid.

El ORB implementa la transparencia de ubicación. Exactamente el mismo mecanismo de peticiones es utilizado por el cliente y el objeto CORBA, sin considerar en dónde está localizado el objeto. Esto podría estar en el mismo proceso con el cliente, localizado en la misma sala o en otro punto del planeta. El cliente no podría notar la diferencia.

El ORB implementa la independencia del lenguaje de programación para la petición. El cliente que hace la petición puede estar escrito en un lenguaje de programación diferente desde la implementación del objeto CORBA. El ORB hace la traducción necesaria entre los lenguajes de programación. Las asociaciones de los lenguajes están definidas para todos los lenguajes de programación populares.[56]

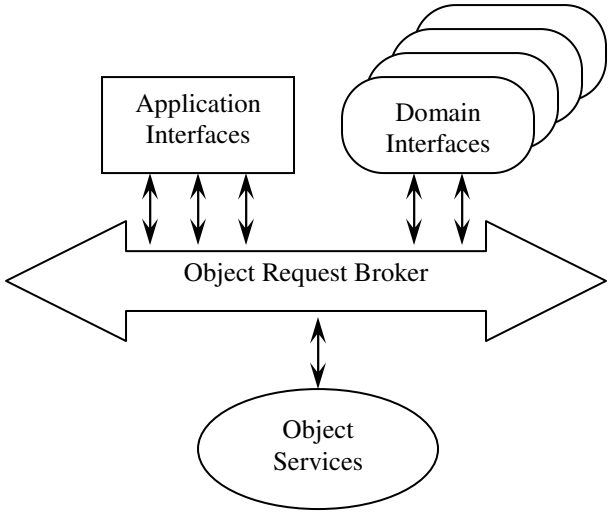
Al ser ORB un servicio distribuido, se podría suponer que tenemos varias versiones de CORBA ORB y esto es muy cierto. CORBA es una especificación. Es una guía para implementar productos. Muchos vendedores proveen productos CORBA para varios lenguajes de programación. Los productos CORBA que soportan al lenguaje de programación Java más comunes son los siguientes[57]:

ORB	Description
The Java 2 ORB	El ORB de Java 2 está incluido dentro del SDK de Java2 provisto por Sun. Muchas características están perdidas.
VisiBroker for Java	Un ORB popular para Java de Inprise Corporation. VisiBroker también se encuentra incrustado en otros productos. Por ejemplo, es el ORB que está incluido en el navegador Netscape Communicator.
OrbixWeb	Un ORB popular de Java propiedad de Iona Technologies.
WebSphere	Un servidor de aplicaciones popular con un ORB de IBM.
Netscape Communicator	Los navegadores de Netscape tienen una versión de VisiBroker incrustados en ellos. Los Applets pueden realizar solicitudes en objetos CORBA sin descargar clases de ORB en el browser, puesto que ya se encuentran allí.
Various free or shareware ORBs	Implementaciones de CORBA para varios lenguajes están disponibles para descarga en la Web de varias fuentes.

Tabla 2 Productos CORBA que soportan al lenguaje de programación Java

[56] Ibid., p. 4
[57] Ibid., pp. 5, 6

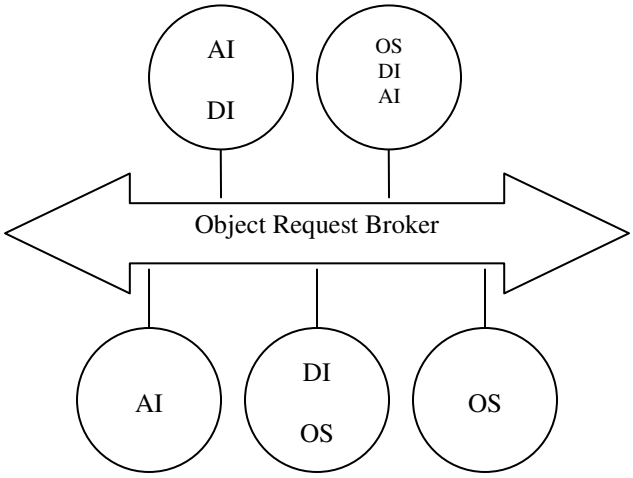
El ORB maneja ciertas categorías de interfases. Son tres principalmente[58]:



- **Object Services** son interfaces de dominio independiente, orientadas horizontalmente, utilizadas por muchas aplicaciones de objetos distribuidos. Por ejemplo, OMG Naming Service y OMG Trading Service para permitir a las aplicaciones buscar y descubrir referencias de objetos.
- **Domain Interfaces** juegan roles similares a los Object Services excepto que las Domain Interfaces son específicas del dominio, o verticalmente orientadas. Por ejemplo, hay Domain Interfaces utilizadas en aplicaciones del cuidado de la salud que son únicas para esa industria, como Servicio de Identificación de Personal. Otras interfaces son específicas para finanzas, manufactura, telecomunicaciones, y otros dominios. Se tiene multiplicidad de dominios.
- **Application Interfaces.** Son desarrolladas específicamente para una aplicación dada. No están estandarizadas por la OMG. Sin embargo, si ciertas interfaces de aplicación empiezan a aparecer en muchas aplicaciones diferentes, llegan a ser candidatas para estandarización en una de las otras categorías de interfaz.

Ilustración 8 Categorías de interfases del ORB

Estas categorías de interfases están relacionadas con el concepto de OMG Object Framework.59



AI=Application Interfaces
DI=Domain Interfaces
OS=Object Services

El concepto de Object Framework, se define a partir de las categorías de interfaces que usa el ORB.
Reconoce y promueve la noción de programas basados en CORBA, los cuales están compuestos de **componentes multiobjetos que soportan una o más de las categorías de interface del OMA**. La figura representa tales componentes como círculos.
El término framework en este contexto es utilizado siguiendo la definición clásica del marco de trabajo de software: una solución parcial de un conjunto de problemas similares que requiere la personalización de dicha aplicación para ser completada.

Ilustración 9 Definición de Object Framework

Una de las categorías de interfases define un conjunto de servicios distribuidos que soportan la integración y la interoperabilidad de los objetos distribuidos. Como se muestra en la gráfica de abajo, los servicios conocidos como CORBA

[58] Ibid., p. 12
[59] Ibid., p. 13

Services , OS o COS, son definidos en la primer capa del ORB. Esto es, estos son definidos como objetos CORBA estándar con interfases IDL, algunas veces referidos como "Object Services"[60]

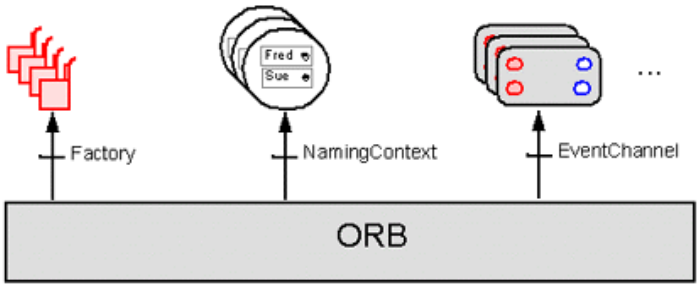


Ilustración 10 Algunos servicios de objetos CORBA

En la gráfica anterior se describen solamente tres de los CORBA services. ¿Qué otro tipo de CORBA services existe? En la siguiente tabla se muestran todos los tipos de servicios que se pueden encontrar[61]:

Service	Description
Object life cycle	Define cómo los objetos CORBA son creados, eliminados, movidos y copiados.
Naming	Define cómo los objetos CORBA pueden tener nombres simbólicos amigables.
Events	Separa la comunicación entre objetos distribuidos.
Relationships	Provee relaciones arbitrarias de tipo n-ario entre objetos CORBA.
Externalization	Coordina la transformación de objetos CORBA y de y hacia medios externos.
Transactions	Coordina el acceso atómico a objetos CORBA.
Concurrency Control	Provee un servicio de bloqueo para objetos CORBA a fin de garantizar acceso serial.
Property	Soporta la asociación de pares nombre-valor con objetos CORBA.
Trader	Soporta la búsqueda de objetos CORBA basándose en propiedades que describan el servicio ofrecido por el objeto.
Query	Soporta consultas sobre objetos

Tabla 3 Servicios CORBA para la administración de objetos.

¿Cómo pueden estos servicios ayudar al cliente de CORBA **a obtener una referencia a Objetos**? Las referencias a objeto son la única vía por la que el cliente puede alcanzar objetos destino. Un cliente no puede comunicarse al menos que retenga la referencia a objeto. Para que el cliente obtenga las referencias existen algunos métodos. Por ejemplo, el servidor puede…

… regresar una referencia como el resultado de una operación (como el valor de retorno o como un parámetro inout o bien out)
… notificar una referencia en algún servicio bien conocido, tal como el Naming Service o el Trading Service.
… publicar una referencia a objeto al convertirla en string y escribirla en un archivo.
… transmitir la referencia a objeto por algún otro mecanismo fuera del canal de comunicación, por ejemplo enviando un e-mail o publicándola en una página Web.

La forma más común para que el cliente adquiera una referencia a objeto es recibirla en la respuesta de la invocación de la operación. En tal caso, las referencias a objetos son los valores de los parámetros y no son diferentes de algún otro tipo de valor, como un string. Los clientes simplemente contactan al objeto, y el objeto regresa una o mas referencias. En este caso, los clientes pueden navegar en una "object Web" muy similarmente a como alguien puede seguir las ligas de hipertexto.

[60] MAGELANG. Op. Cit., p.5
[61] Ibid.

Los clientes raramente usan otros métodos para adquirir las referencias a objetos. Por ejemplo, la búsqueda de una referencia en el Trader o leer una referencia de objeto de un archivo típicamente pasa solamente durante el arranque de la aplicación. Después que el cliente tiene las primeras referencias a objetos, usa esas pocas para adquirir más referencias a otros objetos al invocar operaciones.

Sin tomar el consideración el origen de las referencias a objetos, estas son siempre creadas por el ORB run time en representación del cliente. Esta aproximación oculta la representación interna de las referencias del cliente[62].

Al haber analizado los actores, los tipos de mensajes o "requests" en CORBA, y los mecanismos del manejo de objetos y referencias, el tema llega a desembocar en la pregunta. ¿Qué protocolos de transporte en CORBA son utilizados para establecer la comunicación entre objetos? GIOP e IIOP son los protocolos de transporte de CORBA. CORBA 2.0 trajo consigo una arquitectura de interoperabilidad entre ORBs llamado General Inter-ORB Protocol (GIOP). GIOP es un protocolo abstracto que especifica la sintaxis de transferencia y el conjunto de formatos de mensaje estándar para permitir que ORBs desarrollados de manera independiente se puedan comunicar sobre algún transporte orientado a conexión. Por otra parte, Internet Inter-ORB Protocol (IIOP) especifica cómo GIOP se debe implementar sobre TCP/IP (Transmisión Control PRotocol/Internet Protocol). Todos los ORBs exigen la interoperabilidad de CORBA 2.0, y debe existir una construcción de GIOP e IIOP. Los ORBs contemporáneos reúnen tales requisitos. [63]

Viéndolo más detalladamente, ¿Qué es IIOP y para qué sirve? CORBA 2.0 añadió interoperabilidad como meta en la especificación. En particular, CORBA 2.0 define un protocolo de red, llamado IIOP (Internet Inter-ORB Protocol), que permite a los clientes usar productos CORBA de algún vendedor para comunicarse con objetos de un producto CORBA de algún otro vendedor. IIOP trabaja a través de Internet, o más precisamente, a través de alguna implementación TCP/IP.

La interoperabilidad es más importante en un sistema distribuido que la portabilidad. IIOP es utilizado en otros sistemas que ni siquiera tratan de proveer la API de CORBA. En particular, IIOP es utilizado como el protocolo de transporte para una versión de Java RMI (también llamada RMI sobre IIOP). Desde que los EJB están definidos en términos de RMI, estos también pueden usar IIOP. Varios servidores de aplicación disponibles en el mercado usan IIOP pero no exponen la API de CORBA completa. Debido a que estos usan IIOP, los programas escritos para estas API's diferentes pueden ínteroperar entre ellos y con programas escritos para la API de CORBA.[64]

Por fín, al tener todos los elementos involucrados en la arquitectura de CORBA, ya se puede responder a la pregunta, ¿cuáles son los pasos para construir una aplicación en CORBA? Se tienen al menos 6 pasos:

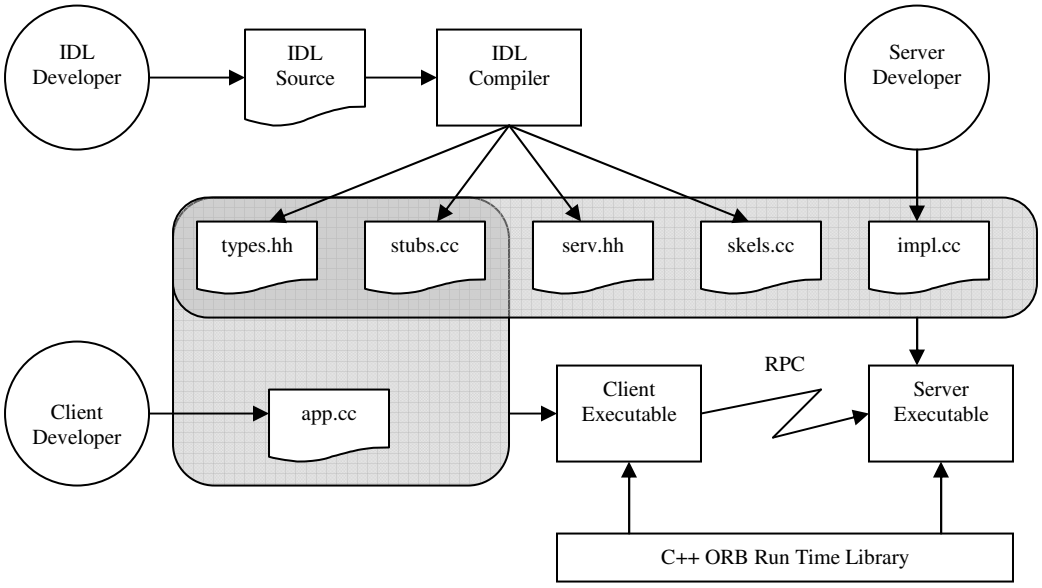
1. **Determinar los objetos de la aplicación y definir sus interfases en IDL**. Así como se desarrolla un programa orientado a objetos, se deben buscar los objetos, definir sus interfases, y definir como ellos están relacionados uno con otro. Este proceso es regularmente dificultoso e iterativo, y CORBA no hace esta parte del ciclo de vida del desarrollo facil para el programador.
   De hecho, para diseñar una aplicación CORBA, o alguna aplicación de objetos distribuidos de ese tipo, es la mayoría de las veces mas difícil que diseñar un programa normal debido a que se debe tratar con los asuntos relacionados a la distribución. Aunque CORBA y sus mapeos de lenguaje ocultan mucha de la complejidad y muchos de los detalles de bajo nivel asociados con la programación típica en red, no considera de forma mágica todos los problemas que se encuentran en los sistemas distribuidos, tales como retardo de mensajes, partición de red y falla parcial del sistema. Basando la aplicación en un ORB ayuda en esa consideración, pero todavía se debe considerar en el diseño los retardos y los modos de la falla distribuida, si se desea escribir aplicaciones distribuidas de alta calidad.
2. **Compilar las definiciones de IDL en stubs y skeletons de C++**. Las versiones de ORB normalmente proporcionan compiladores de IDL que siguen las reglas de mapeo de lengjaue para compilar elIDL en stubs del cliente y skeletons del servidor. Para C++, los compiladores de IDL típicamente emiten archivos de encabezado de C++ que contienen declaraciones para clases proxy, skeletons del servidor y otros tipos soportados. Estos tambien generan archivos de implementación en C++ que construyen las clases y los tipos declarados en los archivos reencabezado.

[62] VINOSKY. Op. Cit., pp. 28, 29
[63] Ibid., pp. 23, 24
[64] MageLang. Op. Cit., p. 4

Para traducir las definiciones de IDL en C++, se genera un código base que permite escribir clientes y sirvientes que respectivamente accedan e implementan objetos CORBA soportando las interfaces IDL.

3. **Declarar y programar clases sirvientes en C++ que incorporen a los objetos CORBA.** Cada uno de los objetos CORBA deben ser encarnados por una instancia de clase sirviente C++ antes que el ORB pueda enviarle requests a ella. Se deben definir las clases sirvientes e implementar sus funciones miembro (las cuales representan los métodos IDL) para llevar a cabo los servicios que se desea que los objetos CORBA provean a los clientes.
4. **Escribir un programa main en el servidor.** Así como en todos los programas en C++, la función main provee los puntos de entrada y salida para una aplicación CORBA C++. Para el servidor, el main debe inicializar el ORB y el Portable Object Adapter (POA), crear algunos sirvientes, permitir que los encarnen los objetos CORBA, y finalmente, empezar a escuchar las peticiones de los clientes.
5. **Compilar y ligar los archivos de implementación del servidor con los stubs y skeletons generados para crear el ejecutable del servidor.** Para un servidor CORBA C++, se debe proveer la implementación de los metodos. Los stubs y skeletons generados proveen la implementación del tipo de IDL y el código de atención de peticiones para traducir los CORBA requests entrantes en llamadas a funciones C++ en los sirvientes.
6. **Escribir, compilar y ligar el código del cliente junto con los stubs generados.** Finalmente, se construyen los clientes para obtener primeramente las referencias a objeto a los objetos CORBA. Para ejecutar los servicios en su representación, los clientes entonces invocan operaciones en los objetos CORBA. El código del cliente invoca las requisiciones y recibe las respuestas como si estuviera haciendo llamadas a funciones C++ normales. Los stubs generados traducen esas llamaas a funciones en invocaciones CORBA requests en los objetos del servidor.[65]

¿Cómo podría ilustrarse un entorno de desarrollo de CORBA utilizando un solo lenguaje de programación, por ejemplo C++? Eso podría hacerse a través del siguiente esquema[66]:



[65] VINOSKY. Op. Cit. pp. 33, 34
[66] Ibid., pp. 55, 56

- **Archivo de encabezado types.hh**. Contiene definiciones que corresponden a los tipos utilizados por el IDL. Esta incluido en el código fuente del cliente y del servidor para asegurar que el cliente y el servidor están de acuerdo acerca de los tipos e interfases usadas por la aplicación.
- **Archivo de encabezado serv.hh**. Contiene definiciones que corresponden a los tipos utilizados en el IDL, pero las definiciones son específicas para el lado del servidor, or lo que este archivo está incluido solamente en el código fuente del servidor.
- **Código fuente stubs.cc.** Provee una API para el cliente para enviar mensajes a los objetos remotos. El código fuente del cliente (app.cc, escrito por el desarrollador del cliente) contiene la lógica de aplicación del lado del cliente. El código del stub y del cliente son compilados y ligados en el ejecutable del cliente.
- **El archivo de skeleton**. Contiene el código fuente que proporciona una interfaz de levantamiento de llamadas del ORB en el código del servidor escrito por el desarrollador y provee la conexión entre la capa de red del ORB y el código de la aplicación. El archivo de implementación del servidor (impl.cc escrito por el desarrollador del servidor) contiene la lógica de aplicación del lado del servidor (las implementaciones del servidor, apropiadamente llamadas sirvientes). El código fuente del skeleton y el stub y el código fuente de implementación son compilados y ligados en el ejecutable del servidor.

Ilustración 11 Entorno de desarrollo de CORBA utilizando un solo lenguaje de programación

En contraste, ¿Cómo se vería la aplicación si los programadores utilizaran diferentes entornos de desarrollo para el Cliente y el Servidor en CORBA?[67]

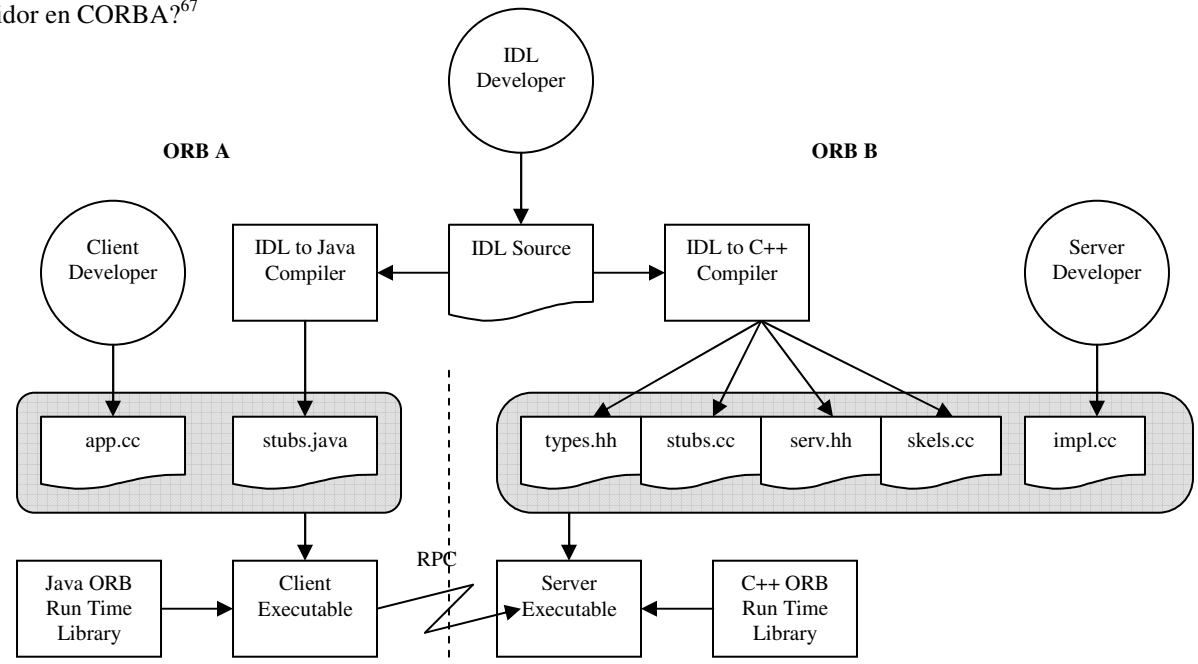


Ilustración 12 Diferentes entornos de desarrollo para el Cliente y el Servidor en CORBA

El cliente y el servidor pueden no compartir alguna fuente o componentes binarios si estos están desarrollados en diferentes lenguajes o diferentes ORBs. Claramente, un cliente escrito en Java puede no incluir un archivo de encabezado C++. Similarmente, compartir el código fuente o los binarios de diferentes vendedores de ORB es imposible debido a que esto podría crear dependencias cerradas en las implementaciones de clientes, servidores y librerías para el tiempo de ejecución.

La figura muestra la situación cuando un cliente escrito en Java es desarrollado con el vendedor del ORB A, y el servidor correspondiente está escrito en C++ y está desarrollado con el vendedor del ORB B. En este caso, los desarrolladores del cliente y el servidor son completamente independientes, y cada uno utiliza su propio entorno de desarrollo, mapeo de lenguaje, e implementación de ORB. **La única liga entre los desarrolladores del cliente y el servidor es la definición de IDL que cada uno utiliza.**

[67] Ibid.

Como comentario final a esta sección, debe decirse que *los sistemas en CORBA son escalables para mejorar su rendimiento* porque no se está limitado a una simple implementación del cliente o el servidor. Por citar un caso, se pueden construir múltiples servidores, cada uno de los cuales implementen las mismas interfases pero usen diferentes implementaciones (esto es, con diferentes características de desempeño). Múltiples implementaciones del servidor pueden coexistir en el mismo sistema. Este arreglo provee un mecanismo de escalabilidad fundamental en CORBA: si se observa que el proceso del servidor empieza a decaer conforme el número de objetos se incrementan, se pueden correr servidores adicionales para las mismas interfases en diferentes computadoras.

Tales **servidores federados** proveen un servicio lógico único que está distribuido sobre un número de procesos en diferentes máquinas. Cada servidor en la federación implementa las mismas interfases pero almacenan diferentes instancias de objetos. Como es de esperarse, los servidores federados deben asegurar de alguna forma la consistencia de alguna base de datos que compartan a través de la federación, posiblemente usando el OMG Concurrency Control Service[68].

Para ilustrar la tecnología de objetos distribuidos concretamente de IDL, nos pareció adecuado probar la tecnología Java IDL para hacer interactuar a los objetos en diferentes plataformas a través de la red. Java IDL permite que los objetos se comuniquen sin importar si fueron escritos en el lenguaje de programación Java, o bien en otro lenguaje tal como C, C++, COBOL, u otros.

Java IDL está basado tambien en el modelo de objetos distribudos definido por estándares industriales llamado CORBA, de las siglas Common Object Request Brokerage Architecture. Una característica clave de CORBA es IDL, un lenguaje de definición de interfaces neutral al lenguaje. Cada lenguaje que soporta CORBA tiene su propio mapeo IDL, y como su nombre establece, Java IDL establece el mapa hacia Java.

Para soportar la interacción entre objetos en programas por separado, Java IDL provee un Object Request Broker, u ORB. ORB es una librería de clase que permite comunicación a bajo nivel entre aplicaciones Java IDL y otras aplicaciones que cumplen con la especificación CORBA.

La forma mas simple de ilustrar las tareas básicas necesarias para construir una aplicación CORBA distribuida es usando Java IDL en la construcción de un simple programa, por ejemplo, el clásico “Hola Mundo”, cuya simple operación regresa una cadena (String) para imprimir.

Alguna relación entre los objetos distribuidos tiene dos partes: el cliente y el servidor. El servidor provee una interfaz remota, y el cliente llama a esa interfaz remotamente. Estas relaciones son comunes para la mayoría de los estándares de objetos distribuidos, incluyendo Java Remote Method Invocation (RMI, RMI-IIOP) y CORBA. En éste contexto, los términos cliente y servidor definen un nivel de objetos en lugar de una interacción a nivel de aplicación. En éste sentido, un objeto simple podría ser el cliente de una interfaz provista por un objeto remoto y al mismo tiempo implementar una interfaz para ser llamada remotamente por otros objetos.

La siguiente figura muestra como un objeto distribuido de un objeto es compartido entre un cliente y un servidor CORBA para construir la clásica aplicación “Hello World”:

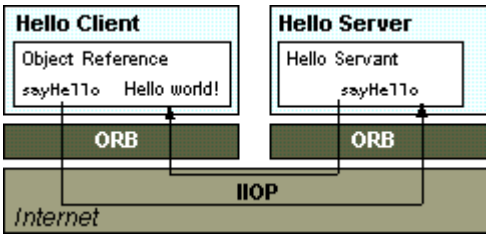


Ilustración 13 Un objeto distribuido de un método compartido en CORBA.

[68] Ibid., p. 54

En el lado del cliente, la aplicación incluye una referencia para el objeto remoto. La referencia de objeto tiene un método de stub, el cual espera a que el método sea llamado remotamente. El stub esta incrustado en el ORB, por lo que la llamada invoca las capacidades de conexión del ORB, el cual reenvía la invocación al servidor.

En el lado del servidor, el ORB usa código de skeleton para traducir la invocación remota en una llamada a algún método de un objeto local. El skeleton traduce la llamada y algunos parámetros a su formato de programación específico, y llama al método que está siendo invocado. Cuando el método regresa, el código skeleton transforma los resultados o errores, y los envía de regreso al cliente a través de los ORBs.

Entre los ORBs, la comunicación procede mediante un protocolo compartido, IIOP, el protocolo Internet Inter ORB. Éste protocolo está bacado en el protocolo de Internet estándar TCP/IP, y define como el ORB que cumple con CORBA pasa la información de regreso. Tal como CORBA e IDL, el estandar IIOP fueron definidos por OMG, el Object Management Group.

Sin subestimar el diseño simple, el programa Hello World permite aprender y experimentar con todas las tareas requeridas para desarrollar casi cualquier programa CORBA que usa invocación estática. Los siguientes pasos proveen una guía general para diseñar y desarrollar una aplicación de objetos distribuidos con Java IDL:

1. **Definir la interfaz remota**. Debe definirse la interfaz para el objeto remoto usando Interface Definition Language IDL de OMG. Aquí, se utiliza IDL en lugar del lenguaje java porque el compilador `idlj` establece un mapa de IDL, genera todos los archivos fuente de stub y skeleton, a través del código de infraestructura para conectarse al ORB. Tambien, usando IDL, es posible para los desarrolladores implementar clientes y servidores en algún otro lenguaje que también cumpla con la implementación Java. Si se está implementando un cliente para un servicio CORBA existente, o un servidor para un cliente existente, podrían obtenerse las interfaces IDL desde el implementador, tal como el proveedor o el vendedor de servicios. Podría entonces ejecutarse el compilador `idlj` sobre ésas interfases y seguir estos pasos.
2. **Compilar la interfaz remota**. Cuando es ejecutado el compilador `idlj` sobre el archivo de definición de interfaz, genera la versión Java de la interfaz, así como los archivos de clase codificados para los stubs y skeletons que permiten que las aplicaciones se enganchen al ORB.
3. **Implementar el servidor**. Una vez que se utilizó el compilador `idlj`, se pueden usar los skeletons generados para unir la aplicación del servidor. Además de implementar los métodos en la interfaz remota, el código del servidor incluye un mecanismo para iniciar el ORB y esperar por la invocación de un cliente remoto.
4. **Escribir el programa cliente**. Similarmente, se utilizan los stubs generados por el compilador `idlj` como la base de la aplicación cliente. El código del cliente es construido sobre los stubs para iniciar su ORB, buscar el servidor usando en nombre del servicio provisto por Java IDL, obtener una referencia para el objeto remoto, y llamar a su método.
5. **Iniciar la aplicación**. Una vez implementado el cliente y el servidor, puede iniciarse el servicio de nombres, iniciar el servidor y finalmente arrancar el cliente [69].

En el **Anexo 10** "Ejemplo de desarrollo en Arquitectura CORBA" mostramos el resultado de efectuar cada uno de éstos pasos para la construcción del programa "Hello World" en su versión CORBA distribuida.

Al haber analizado los conceptos involucrados en el modelo de CORBA y los ejemplos de desarrollo de aplicaciones en tal arquitectura, podemos decir que ésta metodología no solamente es nuestra primera opción de desarrollo de sistemas distribuidos, sino que dicha es la antecesora de todas las arquitecturas de objetos distribuidos, lo que la convierte en el principal marco de referencia para otras opciones que tienen como propósito crear especificaciones o series de lineamientos estandarizados para comunicar componentes distribuidos.

[69] Sun Microsystems. CORBA Technology and the Java™ 2 Platform, Standard Edition. USA, CA. 2001 En: Java Documentation :\j2sdk1.4.2\docs\guide\corba\index.html

2.4.2 Protocolos propietarios: RMI

A continuación se expondrá la solución que ha dado Sun Microsystems de Java para comunicación entre objetos distribuidos. Dicha es la arquitectura Java Remote Method Invocation (RMI) la cual ha sido el soporte de comunicación entre objetos distribuidos desde que Java nació. RMI fue originalmente distribuido de manera separada del JDK 1.0 y después fue integrado con el JDK 1.1. RMI ahora ha crecido en riqueza de características y todavía permanece elegantemente simple para los desarrolladores de Java para crear rápidamente aplicaciones distribuidas en formato cliente/servidor. Los paradigmas RMI y CORBA están también mezclándose en las APIs RMI/IIOP y la arquitectura envuelve a ambas.[70]

A pesar que RMI ha sido diseñado para crear sistemas de objetos distribuidos, un requisito de esta arquitectura es que dichos objetos estén compilados en Java. La plataforma Remote Method Invocation es un modelo de comunicaciones de objetos distribuidos centrado en Java. Al usar la infraestructura y los "packages" de RMI, los clientes de Java basados en RMI pueden invocar métodos remotamente, en los objetos servidor basados en Java RMI. RMI es puramente un modelo de comunicaciones de objetos distribuidos. Los métodos de objetos Java son llamados remotamente contra alguna invocación remota de procedimientos funcionales como en el caso con el precursor conceptual de RMI: Remote Procedure Calling (RPC). Los clientes RMI se comunican transparentemente con servidores distribuidos al llamar a métodos del objeto proxy que se encuentra en el lado del cliente. El proxy serializa los parámetros del método enviados a éste y los dirige al representante del servidor distribuido. El representante del servidor distribuido decodifica los parámetros de la línea serial y los pasa a la instancia apropiada de objeto en el servidor distribuido. El método regresa los valores, yendo a través de un proceso similar de ordenamiento-decodificación.[71]

¿Qué conjunto de características reúne RMI para el programador de comunicaciones empresariales distribuidas? RMI permite que entre el cliente y el servidor se transfieran objetos como parámetros, y regresen un valor o una referencia. Si un tipo de clase usado en un parámetro de método o un tipo de datos de retorno es desconocido por el cliente o el servidor, puede ser cargado dinámicamente usando las bases de código de RMI. Aunque los objetos de servidor pueden ser construidos explícitamente y ligados en el lado del servidor para buscar el servidor, los objetos del servidor pueden también ser activados automáticamente sobre la petición del cliente sin alguna instanciación previa de objeto en el lado del servidor. RMI también provee un medio para la recolección de basura en el entorno distribuido, para limpiar cualquier objeto de servidor distribuido que no es utilizado ni referenciado por algún cliente distribuido. En los casos en que los firewalls limitan las conexiones en puertos, RMI puede crear un túnel a través del firewall vía Http. RMI ahora también permite una personalización completa de los tipos de conexión de sockets subyacente, realizada entre clientes y servidores.

Por otra parte, RMI *no solo tiene semejanzas con CORBA, comparten funcionalidades en el entorno Java*. Conceptualmente, la plataforma RMI provee muchos de los mismos roles que CORBA ORB y CORBA Naming Service proveen. Sin embargo, *RMI también tiene la capacidad para pasar objetos de Java por valor y descargar nuevas clases de Java entre clientes y servidores*. Aunque el estándar de CORBA v2.3 ha definido la especificación de Objetos por valor para objetos CORBA, este permanece muy complejo y en algunos casos pierde la especificación definida. De hecho, la forma de reconciliar RMI y CORBA parece residir en tener un amplio manejo de la interoperabilidad en la especificación de Objetos por Valor de CORBA usando RMI/IIOP.[72]

Al comparar las arquitecturas RMI y CORBA, se encuentra que *RMI es una arquitectura más ligera que CORBA*. Transferir objetos Java móviles a través de RMI entre clientes Java y servidores Java es todavía y siempre será considerablemente más fácil de implementar que pasar objetos por valor en CORBA. RMI es también una solución más ligera en peso para aplicaciones Java. Por supuesto, aunque CORBA tiene todavía la interoperabilidad de lenguajes a su favor, este hecho llega a ser menor importante en la medida que más aplicaciones son construidas usando Java. RMI y Java son plataformas propiedad de Sun, mientras que las interfases estándar de CORBA permiten implementaciones de cómputo distribuido de

[70] SICAP. "Tecnologías para Comunicación Remota" En: Arquitectura de aplicaciones empresariales en Java, p. 1
[71] Ibid.
[72] Ibid.

plataforma abierta. Los servicios distribuidos pueden ser construidos libremente completamente en RMI, pero usar la nueva tecnología de servicios de comunicaciones distribuidas JINI no es libre del uso comercial.[73]

Ahora bien, ¿qué descripción puede darse de la arquitectura RMI? La arquitectura RMI básica está descrita en la figura 16.1. Los clientes RMI contactan con un objeto implementando una interfaz Java que corresponda a esas interfases remotas expuestas por un servidor RMI en particular. La interfaz es implementada actualmente por un stub RMI que toma las llamadas del cliente RMI y las compone en paquetes serializados de información que pueden ser enviados sobre el cableado físico. Similarmente, los stubs descomponen la información en la respuesta serializada de los servidores RMI en objetos Java que pueden ser utilizados por el cliente RMI. En el lado del servidor […]un skeleton RMI de algún tipo decodifica las peticiones del cliente en objetos Java, pasa los parámetros a los métodos del servidor y compone respuestas en la forma serializada, ideal para la transmisión sobre el cableado físico[74].

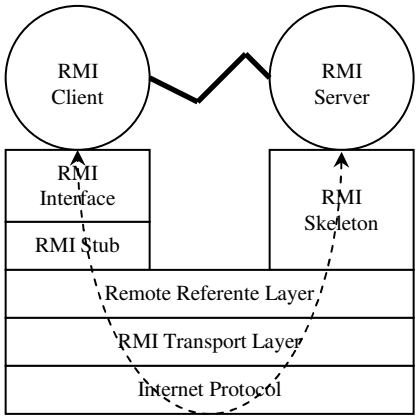


Ilustración 14 La arquitectura RMI

¿Cómo funcionan todos estos elementos de la arquitectura? La capa de referencia remota RMI toma los datos serializados de los RMI stubs y maneja el protocolo de comunicaciones específico de RMI construido por encima del protocolo de transporte. Las labores de la capa de referencia remota incluyen resolver las direcciones de los servidores RMI, inicializar conexiones, y activar los servidores remotos. RMI actualmente soporta dos protocolos generales de mensajería en la capa de referencia. JRMO es un protocolo de mensajería de comunicaciones RMI estándar que ha sido utilizado por RMI desde su origen. El protocolo de mensajería IIOP de CORBA es ahora también posible con la extensión estándar RMI/IIOP

La capa de transporte RMI es responsable de la administración de la conexión y para la provisión de transferencia datos confiable entre puntos finales. El tipo de capa de transporte RMI más ampliamente utilizada es TCP. En ambos casos, JRMP e IOP operan sobre TCP. Sin embargo, RMI v1.2 ha introducido la capacidad de personalizar que tipo de protocolo de transporte subyacente usar, siendo IP aún el protocolo de la red de computadoras.[75]

Entrando en un nivel más de detalle, ¿qué paquetes de Java existen para Implementar RMI? La arquitectura lógica de RMI está dividida en los siguientes conjuntos de clases:

1. **java.rmi** es el paquete central de RMI que contiene la interfaz remota, algunas clases llave, y muchas excepciones estándar.
2. **java.rmi.server** es el paquete central para interfases RMI del lado del servidor, así como clases e interfases.
3. **java.rmi.registry** provee una interfaz y una clase para interactuar con el servicio de búsqueda en RMI**.**
4. **java.rmi.activation** provee las interfases y las clases necesarias para implementar objetos del servidor que pueden ser activados sobre una petición del cliente.

[73] Ibid.
[74] Ibid., p. 2
[75] Ibid.

5. **java.rmi.dgc** provee la interfaz y las clases para utilizar la característica distributed garbage collection de RMI.
6. **javax.rmi** actualmente contiene una clase para implementar objetos portables del tipo RMI/IIOP.
7. **javax.rmi.CORBA** contiene todas las clases específicas de CORBA, e interfases para implemntar y usar objetos RMI/IIOP[76].

En relación con lo anterior, las herramientas de apoyo para implementar aplicaciones y proveer la infraestructura RMI en tiempo de ejecución que provee el fabricante son las siguientes:

- **RMI Compiler (rmic)**. Esta utilería para línea de comando es utilizada para crear archivos de clase Java de stub y skeleton, basándose en una implementación del servidor RMI compilada. Esta herramienta viene equipada con RMI de J2SE/J2EE. Previamente, la versión de esta herramienta que permite la generación de stubs y skeletons en RMI/IIOP fue provista con un componente distribuido por separado RMI/IIOP, pero ahora está integrada con J2SE/J2EE.
- **IDL to Java Compiler (idlj)**. Esta herramienta es utilizada para generar stubs y skeletons ajustables a RMI/IIOP a partir de una definición IDL.
- **RMI Registry (rmiregistry)**. Este comando es utilizado para iniciar un proceso que permite a alguien registrar referencias a objetos servidor RMI/IIOP, usando nombres que pueden ser buscados por clientes RMI distribuidos. Esta herramienta es provista con J2SE/J2EE.
- **RMI/IIOP Naming Service (tnameserv)**. Este comando es utilizado para iniciar un proceso que permite a alguien registrar referencias a objetos servidor RMI/IIOP, usando nombres que pueden ser buscados por clientes CORBA distribuidos. Esta herramienta es provista con J2SE/J2EE.
- **RMI Activation Daemon (rmid)**. Este comando es utilizado para iniciar un proceso que permite a alguien registrar objetos servidor RMI que pueden ser activados (esto es, iniciado y cargado en la memoria) sobre la petición de un cliente. Esta herramienta viene provista con RMI en J2SE/J2EE.
- **HTTP Server.** RMI puede usar un servidor http para descargar dinámicamente clases Java a clientes o servidores que necesiten cargar tales clases en sus ambientes. Aunque algún servidor HTTP puede ser utilizado, RMI requiere solamente la funcionalidad mínima HTTP GET.[77]

¿Cómo funcionan estas herramientas? Antes que un servidor RMI haga sus servicios disponibles para los clientes RMI, una infraestructura en tiempo de ejecución debe ser configurada y puesta en línea por lo que se necesitan seguir los siguientes pasos generales:

- **Configurar e iniciar servidores HTTP**. Deben configurarse e iniciarse servidores http simples en los servidores que proveerán código Java descargable. El código que podría requerir descarga incluye algún código en un lado de la barda distribuida (cliente o servidor) que necesita ser cargada dinámicamente por el otro lado de la barda.
- **Configurar e iniciar el registro RMI**. Se debe configurar e iniciar un servicio de RMI registry el cual activa las referencias a objeto del servidor RMI, referencias dependientes y de las cuales los clientes RMI pueden buscar referencias a objetos distribuidos.
- **Configurar e iniciar un RMI/IIOP Naming Service**. Se debe configurar e iniciar el RMI/IIOP naming service (CORBA Naming) el cual activa las referencias a objeto del servidor RMI, referencias dependientes y de las cuales los clientes RMI pueden buscar referencias a objetos distribuidos.
- **Configurar e iniciar demonios de activación RMI.** Se debe configurar e iniciar un demonio de activación RMI en las computadoras que proveerán código Java activable. Este paso es necesario solo para aquellos servidores RMI que son activables.[78]

Tal como en la sección anterior, al tener definidos todos los elementos de la arquitectura, siempre cabe preguntarse, ¿cuál es el proceso para desarrollar una aplicación utilizando los lineamientos de la arquitectura RMI? Desarrollar una aplicación cliente servidor basada en RMI es por mucho, un proceso directo. Puede darse el caso que se tenga un servicio existente, del cual se requiera proveer una interfaz distribuida, o bien se necesite crear una interfaz distribuida y un nuevo proceso de

[76] Ibid.
[77] Ibid., p. 3
[78] Ibid.

servidor desde el inicio desde las bases. Sin tomar en cuenta esto, se puede seguir los mismos pasos básicos generales para hacer una aplicación RMI:

1. **Definir la interfaz remota**. Primero se debe definir una interfaz Java heredando la interfaz java.rmi.Remote con todos los métodos definidos que desean hacerse distribuidos.
2. **Implementar el servidor RMI** Después se debe implementar la interfaz remota y heredar solo de las clases de objeto del servidor remoto. Los objetos del servidor remoto diferentes pueden ser heredados para proveer un comportamiento preactivado, durmiente pero activable, o bien objeto de servidor RMI/IIOP. Se debe entonces compilar los servidores RMI para usarse en el siguiente paso.
3. **Generar los skeletons y stubs RMI**. Al usar la herramienta rmic, se pueden generar skeletons del servidor RMI para las clases del servidor RMI compiladas. La generación de los skeletons RMI o es necesaria para aplicaciones basadas en RMI v1.2. La herramienta rmic tambien genera los stubs del cliente RMI apropiados. Los stubs RMI v1.2 son creados pasando un –v1.2 como parámetro en rmic.
4. **Implementar un registro del servidor RMI**. Se deberá implementar típicamente una clase por separado que registre un servidor RMI con el RMI registry, registra[79]

Además, ¿qué protocolos de comunicación utiliza RMI para transportar mensajes? RMI es uno de los más importantes paradigmas de comunicación de objetos distribuidos, empleados por las arquitecturas empresariales Java. Inicialmente, RMI utilizaba solamente protocolos propiedad de Sun Microsystems. No obstante, RMI y CORBA están reconciliándose lentamente como paradigmas de comunicaciones interoperables, pero fue en tiempo muy reciente cuando el único modelo subyacente de mensajería disponible para RMI fue el Java Remote Method Protocol (JRMP). JRMP es por mucho, un simple protocolo de transporte en red física que opera sobre TCP/IP. JRMP fue exclusivamente utilizado para RMI en RMI v1.1 y es todavía utilizado por RMI v1.2. Sin embargo, RMI es ahora capaz de operar sobre JRMP o IIOP. No obstante, comprender un poco sobre la composición interna de JRMP arroja luz en RMI para el desarrollador de aplicaciones empresariales, deseando la verdadera compresión de lis límites de RMI/JRMP en cuanto a comunicaciones de clase empresarial, así como sus capacidades[80].

Hablando rápidamente sobre JRMP, se puede observar un formato de Header muy particular. Los paquetes de JRMP enviados entre los puntos de comunicación finales contienen un encabezado de mensaje y uno o más mensajes. El formato del encabezado es simple:

- Cuatro bytes de caracteres ASCII: JRMI
- Dos bytes para el número de versión de protocolo.
- Un byte para identificador de subprotocolo[81].

Este protocolo binario también contiene ciertos identificadores de subprotocolos clasificados como sigue, para indicar el tipo de línea de datos del mensaje que sigue al encabezado del mensaje:

- **SingleOpProtocol**: Indica que sigue un paquete de datos del mensaje.
- **StreamProtocol**: Indica que siguen uno o mas paquetes de datos del mensaje . El cliente RMI y el servidor RMI ofrecen un server socket listener para llamadas entrantes y un socket client endpoint para llamadas salientes.
- **MultiplexStreamProtocol**: Indica que sigue uno o mas paquetes de datos de mensaje. El clienteRMI y el RMI Server multiplexan mensajes entrantes y salientes, sobre una conexión de socket único. Este subprotocolo es típicamente utilizado cuando el cliente RMI está restringido para crear Server socket listeners para manejar llamadas entrantes, por ejemplo, debido a las restricciones de seguridad de los applets.

Como dato notable, al igual que CORBA puede envolver el protocolo GIOP con HTTP, llamándose IIOP, de la misma forma, JRMP puede venir envuelto en el protocolo estándar como HTTP. Para facilitar las capacidades de RMI para crear un túnel a través de servidores HTTP, JRMP permite colocar un identificador HTTPPostHeader para enviar mensajes y un identificador HTTPResponseHeader a los mensajes recibidos. Aparte de estos identificadores, seis tipos clave de mensajes

[79] Ibid., pp. 3,4
[80] Ibid., p. 4
[81] Ibid.

son soportados en JRMP. Tres de los seis tipos de mensajes JRMP pertenecen a mensajes que son la salida de una línea del protocolo RMI desde la perspectiva del cliente:

1. **Call CallData**: Esta forma de mensaje es utilizada para llamadas a métodos remotos. Call data tiene un identificador de objeto utilizado para indicar el objeto destino de la llamada. Un número de operación es utilizado en RMI v11 para identificar la operación destino a invocar. Un número de relleno es utilizado como identificador de versión para verificar que el stub y skeleton que están siendo utilizados son compatibles. Finalmente, cero o más valores de argumento son serializados de acuerdo al protocolo de serialización de objeto que está contenido por los datos de la llamada. Para facilitar la carga de clases dinámica, cierta información de clase está incrustada en las cadenas de datos del mensaje.
2. **Ping**: Prueba para ver si una máquina virtual remota todavía está en operación.
3. **DgAck UniqueID**: Este mensaje es enviado desde el cliente al recolector de basura del servidor distribuido para indicar que las referencias a objeto remoto devueltas de un servidor fueron obtenidas.

Los otros tres tipos de mensajes pertenecen a mensajes recibidos por un cliente que es entrada de una línea de datos de protocolo RMI:

4. **ReturnData ReturnValue**: Esta forma de mensaje es utilizada para regresar resultados de llamadas a métodos remotos. El valor de retorno tiene un byte de retorno que indica si el valor que se regresa es normal o si el resultado es una excepción. Un identificador único es también enviado para identificar al objeto de retorno utilizado por el cliente cuando se notifica al recolector de basura distribuido. Finalmente, una serialización de objeto del valor regresado o una excepción es enviada en el valor de regreso. Para facilitar la carga de clases dinámica, cierta información de clase está incrustada en las líneas de datos de los mensajes.
5. **HTTPReturn**: Este mensaje es enviado como resultado de una invocación en una llamada HTTP.
6. **PingAck**: Este mensaje confirma un mensaje Ping[82].

Una característica más de JRMP es que JRMP soporta mensajes multiplexados, tal como lo hace CORBA. Multiplexar sobre una conexión socket sencilla usando el subprotocolo **MultiplexStreamProtocol** requiere algo de lógica adicional JRMP. Cinco operaciones son definidas para cumplir con el multiplexado. Las operaciones OPEN, CLOSE, y CLOSEBACK manejan la apertura, el cierre, y la confirmación de cierre de la conexión, respectivamente. Las operaciones REQUEST y TRANSMIT son utilizadas para crear una señal cuando los mensajes están siendo intercambiados. Además, debido a que se pueden abrir como límite hasta 65536 conexiones sobre un socket en concreto, un identificador de cuatro bites acompaña cada operación de multiplexado para identificar la conexión virtual en la cual opera. En un lado de la sesión RMI cliente/servidor puede albergar 32768 virtual server socket connections con identificadores en el rango de 0x0000 a 0x7FFF, y el otro lado puede guardar 32768 virtual server socket connections con identificadores en el rango de 0x8000 a 0xFFFF[83].

RMI/IIOP es una extensión Java incorporada como parte de J2SE y J2EE y desarrollada unidamente por Sun e IBM. […]CORBA representa el avance más significativo que nuestra industria ha visto para habilitar el desarrollo de un entorno de comunicaciones estándar en la construcción de sistemas empresariales de software. Con IIOP de CORBA como el protocolo físico de transporte estándar para comunicaciones de objetos distribuidos, la capacidad de RMI para comunicarse sobre IIOP habilita que las aplicaciones RMI puedan ínteroperar con sistemas basados en CORBA. JRMP, por otra parte, fue un protocolo no estandarizado y podía no habilitar las comunicaciones con objetos CORBA de diferente lenguaje. Aunque RMI/IIOP todavía depende de algunas características de JRMP, *RMI/IIOP ahora provee interoperabilidad de comunicaciones basado en Java RMI, con objetos implementados en otros lenguajes*. Los programadores en Java pueden todavía hacer uso de la simplicidad de construcción de aplicaciones RMI, y entonces generar IDL usando el nuevo compilador Java a IDL, provisto también con RMI/IIOP[84].

Al ver tanta diversidad de protocolos, siempre surge la pregunta acerca de cuando usar qué. Pues bien, debido a que Java IDL y RMI/IIOP pueden ser utilizados con J2SE/J2EE, alguien se preguntará qué paquetes utilizar para habilitar las

[82] Ibid., pp. 4, 5
[83] Ibid., p. 5
[84] Ibid.

aplicaciones Java en el entorno de CORBA. La vía principal para pensar en Java IDL es usarla como una forma en que las aplicaciones Java se comunican con clientes y servidores que cumplan con la especificación Java.

- Java IDL y la programación CORBA tradicional será usada principalmente cuando se está creando clientes CORBA basados en Java usando una interfaz IDL predefinida o crear servidores CORBA basados en Java que deben generar un mapeo a un servicio CORBA predefinido.
- RMI/IIOP puede ser utilizado cuando se desea dar importancia a la simplicidad de programación relativa de RMI para crear servidores RMI basados en Java, deseando exponer sus servicios a clientes CORBA con algunas restricciones mínimas de programación[85].

¿En qué ejemplo práctico de RMI puede pensarse? Java RMI permite al programador ejecutar clases de funciones remotas usando la mísma semántica de llamadas a funciones locales de la siguiente manera:

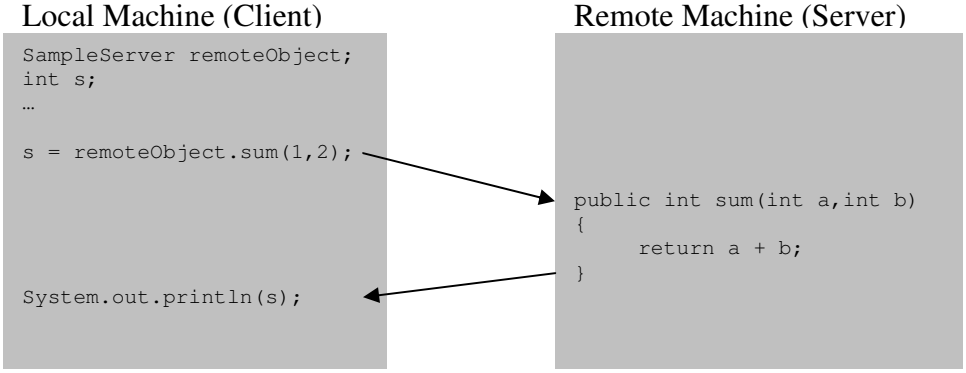


Ilustración 15 Ejecución de clases de funciones remotas usando la mísma semántica de llamadas a funciones locales

Ésta aplicación de objetos distribuidos puede ser construida a través de la arquitectura RMI general:

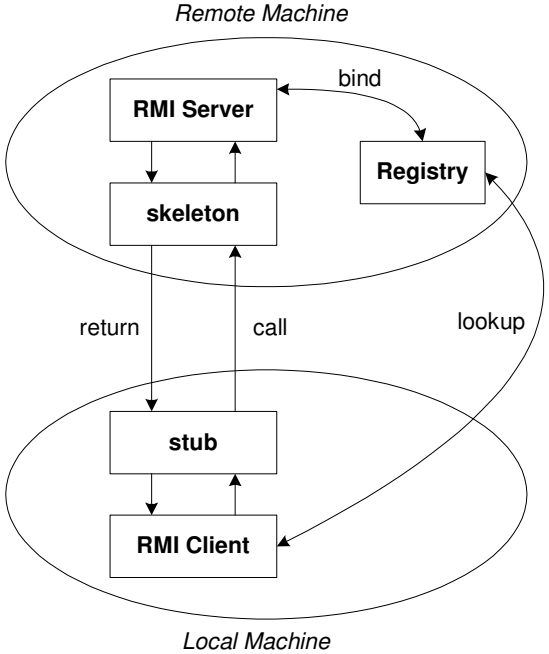


Ilustración 16 Relación de los elementos en la arquitectura RMI

[85] Ibid.

¿Cómo fluye la información en éste diagrama? Básicamente a través de tres pasos.

1. El servidor debe en primer lugar lugar su nombre al registry.
2. El cliente busca el nombre del servidor en el registry para establecer las referencias remotas.
3. El stub serializa los parámetros al skeleton, el skeleton invoca el método remoto y serializa el resultado de regreso al stub.

¿Qué papel desempeña el Stub y el Skeleton en la arquitectura RMI? El siguiente diagrama muestra cómo estos son agentes negociadores de llamadas y respuestas:

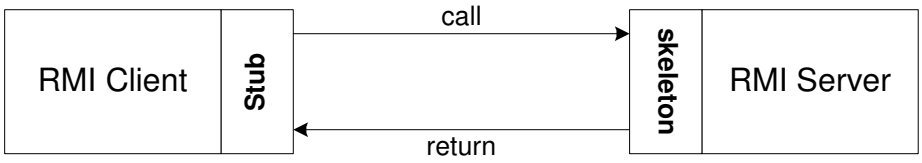


Ilustración 17 Stub y Skeleton son negociadores de llamadas y respuestas

¿Cómo funciona éste modelo? Es a través de cuatro fases:

1. El cliente invoca un metodo remoto, la llamada es reenviada al stub.
2. El stub es responsable de enviar la llamada remota sobre el skeleton que está del lado del servidor.
3. El stub abre un socket al servidor remoto, ejecuta un marshall sobre los parámetros del objeto y reenvia la cadena de datos al skeleton.
4. Un skeleton contiene un método que recibe las llamadas remotas, hace el unmarshall sobre los parámetros, e invoca la implementación de objeto remoto real.

Al haber mencionado la forma en que funciona una aplicación RMI, no resta mas que hacer referencia al **Anexo 11**, "Ejemplo de desarrollo en Arquitectura Java RMI" en donde se muestran los pasos para desarrollar un sistema RMI. Tales pasos son:

1. Definir la interfaz remota.
2. Desarrollar el objeto remoto implementando la interfaz remota.
3. Desarrollar el programa cliente.
4. Compilar los archivos fuente java.
5. Generar los stubs del cliente y los skeletons del servidor.
6. Iniciar el RMI registry.
7. Iniciar los objetos de servidor remotos.
8. Ejecutar el cliente[86].

En resumen, se debe pensar en Java IDL cuando alguien o una circunstancia nos obligue a usar un archivo IDL para hacer un mapa de clientes en servidores distribuidos (esto es, de interfaz a código), y pensar en RMI/IIOP cuando se tienen servidores Java RMI que necesitan exponerse en términos de interfases IDL (esto es, de código a interfaz). Además, se debe ser cuidadoso del hecho que cuando RMI/IIOP es instalado como una extensión estándar a la plataforma J2SE v1.2, éste proveera las bibliotecas Java IDL ORB para trabajar con RMI/IIOP.

2.4.3 Protocolos abiertos: Web Services

Los Web services son un abanico de términos que describen una colección de protocolos estándares de la industria, así como servicios utilizados para facilitar una línea de nivel base para la interoperabilidad entre aplicaciones. El soporte que la industria le ha brindado a los Web services no tiene precedente. Nunca antes tantas firmas tecnológicas habían avanzado de

[86] Vid. http://www.comp.hkbu.edu.hk/~kchui/rmi/rmi.doc

esta forma a fin de soportar un estándar que facilitara la interoperabilidad entre aplicaciones, sin tomar en consideración la plataforma en las cuales están corriendo.[87]

Los Web services son unos de los desarrollos más novedosos en el campo de las ciencias de la computación, y también uno de los menos entendidos. Algunas personas los describen como la tecnología diseñada estrictamente para publicar servicios de software en Internet, mientras que otros piensan que es una arquitectura de propósito general que dispara un cambio fundamental en la forma que los sistemas distribuidos son creados.

Lejos de ser un capricho, los Web services son la evolución lógica del cómputo distribuido. Este cubre todos los términos importantes tales como *SOAP, WSDL y UDDI*, y también aborda todas las preguntas importantes relacionadas con desempeño y confiabilidad.[88]

Para entender la arquitectura Web services primero es indispensable definir a los servicios de software. Un servicio de software es algo que acepta las requisiciones digitales y regresan respuestas del mismo tipo. Utilizando esta definición, una función C, un objeto Java y un procedimiento almacenado escrito en Structured Query Language (SQL) son ejemplos de servicios de software; una aplicación de cómputo que puede ser concebida como un conjunto de servicios bien orquestados.[89]

En base a la definición anterior, puede decirse que los Web services son servicios de software de características determinadas. Hasta ahora, un servicio de software específico puede ser utilizado solamente dentro de un lenguaje o plataforma en particular, y no siempre está disponible a través de la red.

Los Web services son una nueva generación de productos de componentes de software que son independientes del lenguaje, plataforma y ubicación física. *Están construidos en bloques de XML* (Extensible Markup Language) lo que da paso a la siguiente generación de aplicaciones cuyas partes pueden residir en una computadora sencilla o abarcar todo el globo.[90]

Más concretamente, ¿qué significa el término Web services? El término es una abreviatura de "Web of Services", lo que significa que las aplicaciones distribuidas pueden ser ensambladas a partir de una red de servicios de software de la misma forma en que las páginas de Internet son ensambladas a partir de una red de páginas HTML.

La mayoría de las aplicaciones distribuidas serán construidas a partir de Web services, sin tomar en cuenta si han sido desarrolladas para una máquina sencilla, una intranet corporativa o la misma Internet.[91]

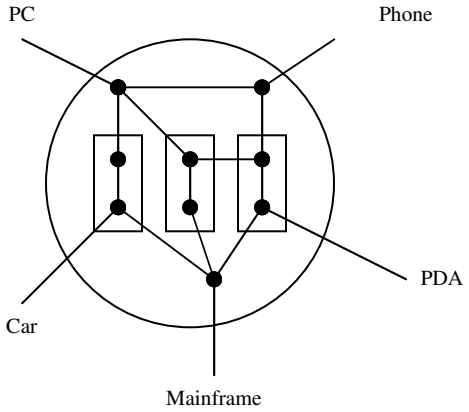


Ilustración 18 En el futuro, los sistemas distribuidos serán una Web de servicios

[87] SHORT, Scott. Op. Cit., p.10
[88] GLASS. Op. Cit., p. 1
[89] Ibid., p. 2
[90] Ibid.
[91] Ibid.

¿Por qué reviste tanta importancia la arquitectura de Web services? Los Web services son importantes porque estos simplifican ampliamente la creación e integración de sistemas distribuidos a gran escala. En la misma forma que HTML amalgama una vasta red de información para uso humano, SOAP, WSDL y UDDI permiten una red de servicios accesibles por computadoras.[92]

¿Qué necesidad originó el concepto de Web services? Internet abrió muchos ojos de las personas al poder del cómputo distribuido, y el negocio empezó a invertir más tiempo en adquirir el poder de las aplicaciones en red. No obstante, **pese a que HTTP y HTML hicieron posible para los consumidores utilizar páginas web remotas, estos no simplificaron la integración de los sistemas de negocios.** Lo que faltaba era una **forma de compartir datos y servicios de software a través de Internet**.

El intercambio de datos fue manejado por XML, una versión más sofisticada de HTML para propósito general, que permite que algún tipo de datos sea representado en una forma simple y portable. Al rodear los datos con etiquetas que indican su significado, los documentos XML son auto descritos, y pueden ser fácilmente manipulados y transformados.[93]

```
<invoice id='13552'>
        <item>football</item>
        <amount>39.95</amount>
</invoice>
```

Ilustración 19 Un documento XML simple

Ahora bien, ¿Cómo se conjugan estos dos mundos, la programación e Internet, aparentemente aislados entre sí? ¿Cómo se transforma, por ejemplo, un componente de sofware en un servicio Web? La forma más fácil para publicar un componente de software como web service es usar un contenedor SOAP, el cual acepte peticiones entrantes, y las atienda con componentes publicados, con traducción automática entre SOAP y la interfaz de lenguaje nativo del componente.

Los contenedores SOAP están disponibles para la mayoría de los lenguajes de programación, incluyendo Java, C++, Perl y C#.[94]

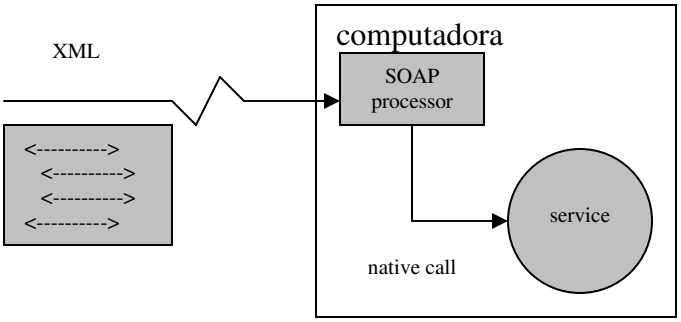


Ilustración 20 Un contenedor SOAP convierte mensajes XML en llamadas locales de la computadora

Antes de continuar, conviene explicar qué es una aplicación de SOAP. SOAP es el nuevo estándar para la comunicación en red entre servicios de software .Es una tecnología de propósito general para enviar mensajes entre puntos finales, y puede ser utilizado para RPC o transferencia de documentos directa. Los mensajes SOAP son representados utilizando XML y pueden ser enviados sobre cualquier capa de transporte. *HTTP es la capa de transporte más común*, con implementaciones también disponibles para Simple Mail Transport Protocol (SMTP), Java Messagin Services (JMS), e IBM MQSeries.[95]

[92] Ibid., p. 7
[93] Ibid., pp. 5,6
[94] Ibid., p. 8
[95] Ibid., pp. 7,8

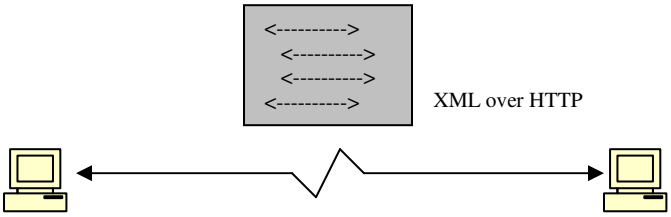


Ilustración 21 Los mensajes SOAP son documentos XML normalmente enviados sobre HTTP

Los dos diagramas anteriores sugieren que esta arquitectura es algo semejante a la arquitectura cliente servidor. En este caso, se tiene un cliente de SOAP y un Web service que también procesa XML, pero ¿Cuál es el funcionamiento de un Cliente SOAP?

Una vez que el componente ha sido publicado como un Web service, algún cliente habilitado para SOAP que conozca la dirección en la red del servicio y los mensajes que este entiende, puede enviar una petición SOAP y obtener una respuesta SOAP. *Para obtener la dirección y la información del mensaje, los clientes SOAP leen un archivo WSDL que describe el servicio Web.*

Afortunadamente, la mayoría de los contenedores SOAP generarán automáticamente un WSDL para los servicios Web que almacenan, por lo que los desarrolladores no tienen que escribir los WSDL manualmente al menos que realmente deseen hacerlo. Una vez que el archivo WSDL es leído, el cliente puede iniciar el envío de mensajes SOAP al servicio Web.[96]

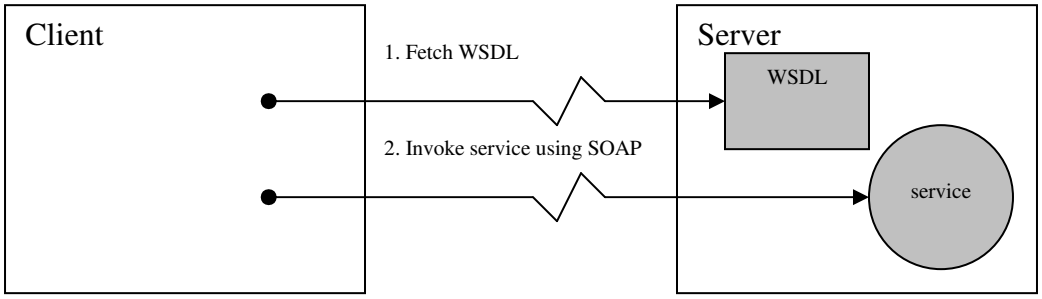


Ilustración 22 Un cliente necesita el WSDL antes de invocar el servicio

Cabe destacar que *algunas implementaciones de PROXY ocultan los detalles de implementación SOAP*. El proceso de binding regresa un PROXY que implementa una interfaz Java cuyos métodos son un reflejo de los existentes en el servicio remoto. Un mensaje enviado al proxy es automáticamente convertido en una petición SOAP, enviada a través de la red, y la respuesta SOAP es convertida de regreso en un resultado Java regular.[97]

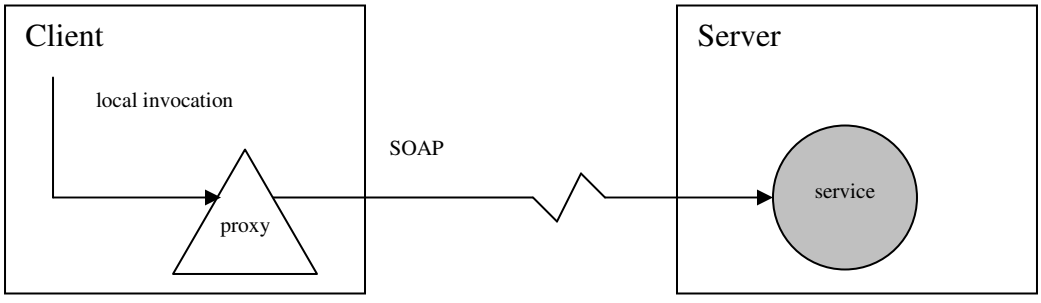


Ilustración 23 Un proxy en el cliente oculta los detalles de comunicación en la aplicación

[96] Ibid., pp. 8, 9
[97] Ibid., p. 12

Al comparar las ilustraciones 22 y 23 podemos ver claramente que el PROXY oculta los detalles de construcción del servicio Web, por lo que, incluso tareas tan triviales como leer un WSDL, escribir un mensaje SOAP, mandarlo al servicio, y obtener el mensaje SOAP de respuesta, pueden ser automatizadas.

¿De cuántas maneras puede programarse un Web service? Es fácil delimitar la respuesta a esta pregunta si partimos del hecho que los SOAP requests siempre son procesados por un servlet, un CGI o un programa en el servidor Web. **Un SOAP request es típicamente aceptado por un servlet**, un CGI o un demonio standalone corriendo en el servidor web remoto. Cuando el servlet obtiene una petición, **este revisa que la petición tiene un campo "SOAPAction"**, y si es el caso, lo envía al contenedor SOAP. **El container usa el URI del POST para buscar el web service objetivo, traducir la carga de datos de XML, y entonces invocar el método en el componente**.[98]

Aquí, no solo la petición se forma con SOAP. La respuesta que otorgan los Web Services también se forman con esta gramática de XML El resultado de la petición es traducida por el contenedor SOAP en una respuesta SOAP y regresada al remitente dentro de la respuesta HTTP. El documento XML es estructurado como el request, excepto que el body contiene el resultado del método codificado. Por convención, el nombre del resultado es igual al nombre del método seguido de la palabra "Response", y el "namespace" del resultado es el mismo que el "namespace" del método original.[99]

¿*Cuáles son los elementos básicos de un servicio Web de Ejemplo*? En su primera fase de adaptación, se puede pensar en que cierta compañia crea un web service que expone un conjunto de operaciones de facturación a crédito a algún cliente SOAP. El web service por si mismo es una delgada capa envolvente que acepta SOAP requests entrantes, las ejecuta en el sistema de crédito de ACME, y regresa el resultado como una respuesta SOAP. ACME también genera un archivo WSDL que describe el eb service, por lo que otros clientes SOAP pueden invocarlo. La compañía hace disponible un WSDL para sus clientes actuales, los cuales quedan satisfechos con esto porque sus sistemas de cómputo pueden ahora realizar facturación a crédito vía SOAP a través de Internet sin tener que involucrar humanos o máquinas de fax.[100]

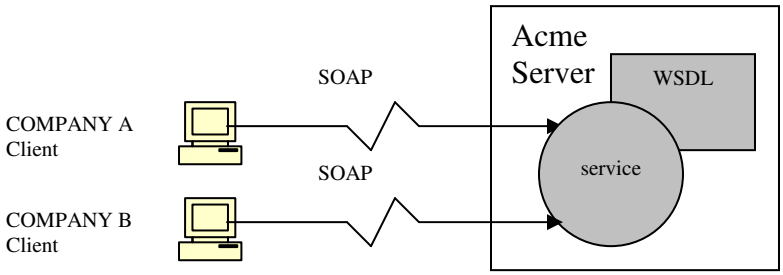


Ilustración 24 Los clientes pueden invocar el servicio de crédito Acme a través de Internet

Si se piensa en la segunda fase de construcción de Web services, se debe pensar en UDDI. Volviendo al ejemplo anterior, en la segunda fase de adopción, la empresa decide hacer su servicio de crédito *disponible para una mayor audiencia*. Ingresa a algún site popular UDDI, obtiene su user/password, e ingresa información general acerca de la compañia, tal como su nombre, razón social, información de contacto, y categoría. Acme también registra su web service de facturación a crédito, incluyendo su dirección URL y una referencia al WSDL.

En este punto, otras compañias pueden localizar al Acme Server a través de su web browser, aprender acerca de sus servicios, y crear clientes SOAP que invoquen sus operaciones de facturación a crédito. Adicionalmente, los clientes SOAP pueden buscar en el registro UDDI usando mensajes SOAP y localizar el servicio de crédito Acme en tiempo de ejecución. Como resultado de hacer su sistema de facturación a crédito disponible como web service, Acme observa que el número de clientes aumenta considerablemente e ingresa a una nueva fase de crecimiento.[101]

[98] Ibid., p. 14
[99] Ibid., p. 15
[100] Ibid., p. 20
[101] Ibid., p. 21

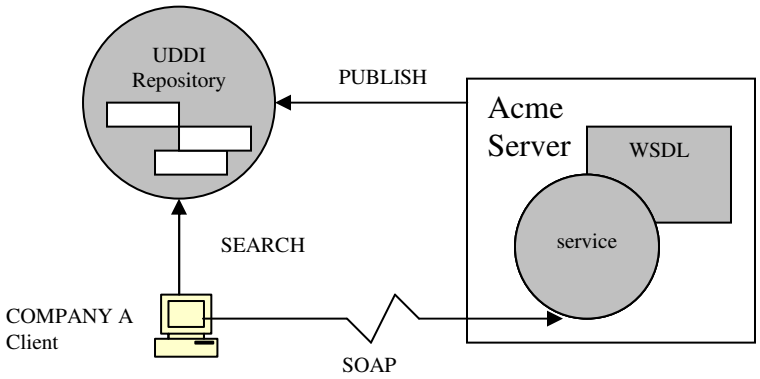


Ilustración 25 UDDI actúa como un punto de reunion para proveedores y consumidores de web services

La tercera fase de implementación de un Web Service corresponde a registrar el servicio en varios UDDI. En la tercera fase de adopción, el servicio de crédito Acme llega a ser muy popular por lo que la especificación de la interfaz WSDL llega a adoptarse por la industria del crédito como un estándar, y otras compañías empiezan a publicar implementaciones competitivas alternativas de la interfaz a UDDI. Aunque esto significa más competencia para Acme, esto también fortalece su posición como el líder intelectual.

Los tres registros públicos UDDI, provistos por IBM, Microsoft y HP, sincronizan su contenido regularmente, por lo que la información ingresada en uno es rápidamente actualizada por los otros. Para ayudar a los desarrolladores en el entendimiento de UDDI, cada compañía también almacena un registro de prueba que está destinado para propósitos educacionales.[102]

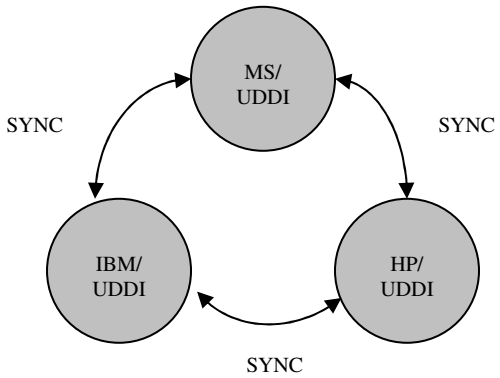


Ilustración 26 Los operadores de UDDI públicos sincronizan su contenido regularmente

Una forma de ver con claridad estas tres fases de construcción es *viendo los bloques de construcción de los Web Services por separado*. Los bloques de construcción centrales necesarios para facilitar la comunicación remota entre dos aplicaciones se describen a continuación:

1. **Discovery (UDDI, DISCO)**. La aplicación cliente que necesita usar la funcionalidad expuesta en un Web service necesita una forma de resolver la dirección del servicio remoto. Esto se lleva a cabo a través de un proceso generalmente definido como discovery. Discovery puede ser provisto a través de un directorio centralizado, así como por otros métodos más adecuados. En DCOM, el Service Control Manager (SCM) provee los servicios de descubrimiento.
2. **Description (WSDL, XML Schema, DOCS)**. Una vez que la dirección para un servicio Web ha sido resuelta, el cliente necesita la suficiente información para interactuar apropiadamente con éste. La descripción de un Web

[102] Ibid.

service contiene metadatos estructurados acerca de la interfase que se pretende consumir por la aplicación del cliente, así como la documentación escrita acerca del Web service, con ejemplos de uso. Un componente DCOM expone metadatos estructurados acerca de sus interfases a través del "type library" (typelib). Los metadatos dentro del typelib de los componetes son almacenados en un formato binario propietario, y son utilizados a través de la interfase de programación propietaria de la aplicación (API)

3. **Message Format (SOAP)**. A fin de intercambiar datos, el cliente y el servidor deben acordar en una forma común de codificar y formatear los mensajes. Una forma estandarizada de codificar los datos asegura que los datos codificados por el cliente deben ser interpretados apropiadamente por el servidor. En DCOM, los mensajes enviados entre el cliente y el servidor están formateados como se define por el protocolo DCOM Object RPC (ORPC).
   Sin una forma estándar de representar los mensajes, desarrollar una herramienta para separar al desarrollador de sus protocolos subyacentes es casi imposible. Al crear una capa de abstracción entre el desarrollador y los protocolos permite que el desarrollador se concentre más en el problema de negocios, y menos en la infraestructura requerida para implementar la solución.
4. **Encoding (XML)**. Los datos transmitidos entre el cliente y el servidor necesitan codificarse en el cuerpo del mensaje. DCOM usa un esquema de codificación binario para serializar los datos contenidos por los parámetros intercambiados entre el cliente y el servidor.
5. **Transport(HTTP,SMTP, and so on).** Una vez que el mensaje ha sido formateado y los datos han sido serializados en el cuerpo del mensaje, el mensaje debe ser transferido entre el cliente y el servidor sobre algún protocolo de transporte. DCOM soporta un número de protocolos propietarios ligados a un número de protocolos de red tales como TCP, SPX, NetBEUI, y NetBIOS sobre IPX.[103]

Quien desarolla sus sistemas distribuidos en el paradigma de Web services dispone de importantes ventajas. Uno de los factores que contribuyen al éxito de los Web services es que están **construidos en los estándares de Internet existentes tales como XML y HTTP**. Como resultado, algún sistema capaz de precompilar texto XML y comunicarse a través del protocolo estándar de transporte de Internet puede comunicarse con un Web service. Las compañías pueden también apoyar sus inversiones que ya han hecho en esas tecnologías.[104] El desarrollador de CORBA o RMI no es tan afortunado en este aspecto, requiere más que solamente utilizar una librería de SOAP: necesita un compilador de IDL, un compilador de Java, paquetes, clases, utilerías, codificación adicional, etc.

Por otra parte, el WSDL es el equivalente en XML de un resumen –este describe que puede hacer el Web service, donde reside, y como invocarlo. Si a usted está familiarizado con CORBA o DCOM, piense que el WSDL es el equivalente en web services del Interface Definition Language (IDL) y las bibliotecas de tipos.[105]

Una de las ideas centrales detrás de los web services es que **las aplicaciones del futuro serán ensambladas a partir de un conjunto de servicios diseñados para ejecutarse en red**. Tan pronto como dos servicios equivalentes pueden identificarse por si mismos a la red en una forma estándar y neutral, una aplicación podría teóricamente escoger entre los servicios competentes alternativos basados en criterios tales como precio o rendimiento. Adicionalmente, algunos servicios podrían ser copiados entre máquinas, permitiendo que una aplicación mejore su rendimiento al instalar los servicios dinámicamente en su almacenamiento local.

Los documentos WSDLs pueden ser vistos como contratos electrónicos del servicio Web. Si usted piensa acerca de ello, el manejo de los WSDL es similar a como el mercado de trabajo humano funciona. Las fuentes de trabajo y compañías de reclutamiento proveen el servicio de contactar trabajadores y empleadores, utilizando curricula y descripciones del puesto para facilitar el proceso de contratación. Si se encuentra un buen prospecto, las partes interesadas intentan negociar términos aceptables. Si se alcanza un acuerdo, el trabajador se mueve a las instalaciones físicas del patrón, o trabaja a distancia para la compañia.[106]

[103] SHORT, Scott. Op. Cit., p. 4
[104] Ibid., p.10
[105] GLASS. Op. Cit., p. 18
[106] Ibid.

En el ejemplo se tocó a un elemento más: Los servicios UDDI. UDDI es un nuevo estándar que permite que la información sea electrónicamente publicada y utilizada, información sobre el negocio y los servicios. La información publicada es almacenada en uno o más registros UDDI, los cuales pueden ser consultados a través de un web browser o por SOAP.[107]

Los Web services, por muchas razones, cuentan con aceptación unánime. *La convergencia de factores tales como HTTP, WSDL, SOAP, XML, y UDDI posibilitan el mismo tipo de efecto en red que permitió HTML, para llegar a ser rápidamente el estándar a la mano para compartir información en la red*. Las tecnologías de cómputo distribuido previas tales como CORBA y DCOM nunca han tenido el beneficio de la aceptación unánime.[108]

Incluso viéndolo por el lado monetario, es más económico desarrollar en arquitectura de Web Services. *Es factible construir aplicaciones distribuidas de buena calidad en una fracción del costo de las aproximaciones previas*.

Por otra parte, se tienen cuatro ejemplos de uso de los web services en acción:

- **Internet/B2B**
- **Internet/B2C**
- **Intranet.** Un registro UDDI privado permite que aplicaciones internas encuentren y usen estos servicios para usar y coordinar los procesos de la compañía, y solo determinados servicios web son expuestos para habilitar la integración con sistemas remotos de terceros.
- **Local Area Network.**[109]

¿Qué características tienen los SOAP Web Services? Estas son:

- **Capacidad de Operación**. Las implementaciones más rápidas de SOAP típicamente obtienen al menos 500 mensajes/segundo en una PC de escritorio de 600Mhz cuando el cliente y el servidor están en diferentes programas en la misma máquina, y alrededor de 300 mensajes/segundo en una fast LAN.
- **Arreglos, objetos y otros tipos de estructuras de datos** complejos pueden ser enviados a través de la red en una plataforma y de forma neutral al lenguaje.
- Los SOAP headers soportan s**eguridad, transacciones y ruteo**.
- Los **tipos de codificación personalizados** pueden ser definidos.
- SOAP soporta operaciones request**-response, one-way, solicit-response y notification-operations**.[110]

Hasta este punto se han expuesto las tres arquitecturas por separado. La siguiente sección hará una comparación de sus características que son similares, evaluará la mejor opción y justificará su selección.

[107] Ibid., p. 19
[108] Ibid., p. 23
[109] Ibid.
[110] Ibid., pp. 17,18

2.5 Selección del uso de la tecnología de Web Services y Justificación

Al recordar los objetivos primarios de las arquitecturas de objetos distribuidos, en primer lugar se necesita alcanzar la interoperabilidad de los sistemas. ¿Cómo fue que **la interoperabilidad se convirtió en una necesidad**? Fue debido al desarrollo histórico de diversas arquitecturas por separado.

Después que los lenguajes de procedimiento llegaron a convertirse en lenguajes funcionales como LISP y lenguajes orientados a objetos como C++, los cuales proveyeron todavía altos niveles de abstracción y permitieron a los desarrolladores crear objetos de software que asemejaban sus partes correspondientes del mundo real. […]Un objeto es todavía otro ejemplo de un servicio de software, solo que esta vez encapsula su comportamiento y provee polimorfismo.

Los nuevos protocolos de cómputo distribuido fueron diseñados para permitir que los objetos se comuniquen en forma más natural que RPC, y el más exitoso fue Common Object Request Broker Architecture (CORBA) y Distributed Common Object Model (DCOM). Un consorcio de grandes compañias soportó CORBA, incluyendo IBM, Oracle y SUN, mientras que DCOM llegó de Microsoft. La mayoría de los desarrollos tempranos de estas tecnologías fueron en ambientes homogéneos y cerrados.[111]

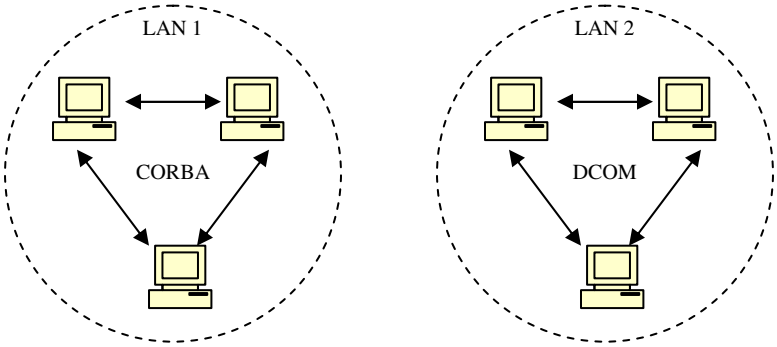


Ilustración 27 Dos LAN, una usando CORBA, y otra usando DCOM

Eventualmente se tuvo la necesidad de **compartir datos entre tales ambientes homogéneos y aislados**, tarea que resultó difícil debido a la *incompatibilidad de tipos de datos y otras características entre ellos*. De hecho, sin considerar sus méritos técnicos relativos, la cosa más molesta para los desarrolladores fue que CORBA y DCOM eran incompatibles, y era un duro trabajo hacer que un protocolo se comunicara con otro. Si se deseaba ligar sistemas que utilizaran protocolos diferentes, típicamente se requería instalar convertidores de protocolo en cada puerta de salida de datos.[112]

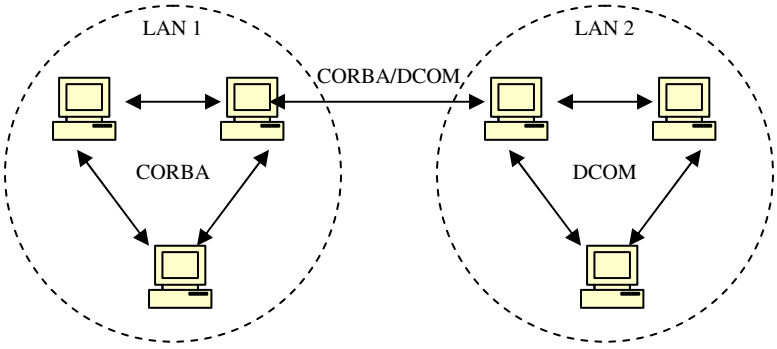


Ilustración 28 Dos LAN, conectadas usando un puente CORBA-DCOM

[111] Ibid., p. 4
[112] Ibid., p. 5

Esos convertidores de protocolo representaban un trabajo adicional para el grupo de desarrollo. Por tal razón la industria se vió en la necesidad de acudir a un estándar diferente, y XML llegó a ser la respuesta de tal problema porque permitió la publicación, localización y descripción de servicios de software.

La publicación de servicios […] es atacada por un nuevo conjunto de estándares llamado Service Oriented Architecture Protocol (SOAP), Web Services Description Language (WSDL) y Universal Description, Discovery and Integration (UDDI). **Construidos sobre HTTP y XML, estos estándares permiten que los servicios de software sean publicados, localizados y utilizados en una forma que es independiente del lenguaje, plataforma y ubicación geográfica**. Los bloques de construcción de software creados con esta tecnología son llamados Web services.[113]

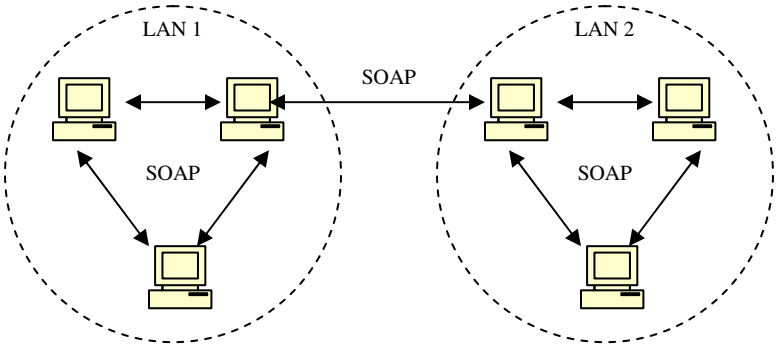


Ilustración 29 SOAP es un protocolo universal para conectar todo

En este sentido los SOAP Web services son un fenómeno histórico notable, porque **la evolución de los Web services revolucionaron los sistemas distribuidos**. Representaron un adelanto tecnológico importante al grado que son el punto de lanzamiento para la construcción de los sistemas en gran escala. [114]

Además de que los XML Web services permiten la convivencia de diversos sistemas, esta tecnología tiene otras características que llaman la atención, entre ellas:

- **Interoperabilidad**. El servicio remoto debe ser capaz de ser consumido por clientes de otras plataformas**.** Además, Cualquier servicio Web puede interactuar con algún otro servicio Web. Gracias a SOAP, la agonía de transferir datos de CORBA a la arquitectura DCOM, y otros protocolos se terminó. Y debido a que los web services pueden ser escritos en cualquier lenguaje de programación, los desarrolladores no necesitan cambiar ambientes a fin de producir o consumir web services.
- **Amigable con Internet**. La solución debe trabajar bien para soportar clientes que utilicen el servicio remoto desde Internet.
- **Interfases con tipos de datos fuertemente definidos.** No debe existir ambigüedad en el tipo de datos enviados a y recibidos de el servicio remoto. Aún más, los tipos de datos definidos por el servicio remoto deben mapear de forma razonable a los tipos de datos por la mayoría de los lenguajes de programación estructurada.
- **Habilidad para apoyarse en los estándares de Internet Existentes**. La implementación del servicio remoto debe descansar sobre los estándares de Internet existentes tanto como sea posible y evitar reinventar soluciones a problemas que ya han sido resueltos. Una solución construida en los estándares de Internet mundialmente adoptados puede apoyarse en los productos y herramientas existentes creadas para dicha tecnología.
- **Soporte para algún lenguaje**. La solución no debe estar fuertemente unida a un lenguaje de programación en particular. Java RMI, por ejemplo, está fuertemente fusionada al lenguaje Java. Podría ser difícil invocar la funcionalidad en un objeto remoto de Java con Visual Basic o Perl. Un cliente debe poder implementar un nuevo Web service o utilizar un Web service existente sin considerar el lenguaje de programación en el cual el cliente es programado.

[113] Ibid., p. 7
[114] Ibid., p. 3

- **Soporte para cualquier infraestructura de componentes distribuida.** La solución no debe estar fuertemente unida a una infraestructura de componentes en particular. De hecho, no debe obligarse a nadie a comprar, instalar o mantener una infraestructura de objetos distribuida solamente para explotar un nuevo servicio remoto o consumir un servicio existente. Los protocolos subyacentes deben facilitar un nivel de comunicación básico entre las infraestructuras de objetos distribuidos existentes tales como DCOM o CORBA.[115]
- **Omnipresencia.** Los web services se comunican entre sí usando HTTP y XML. Cualquier dispositivo que soporte estos estándares puede almacenar y utilizar los web services. Muy pronto, estos estarán presentes en teléfonos, computadoras de automóvil y aún en máquinas de refrescos. Si el inventario de una máquina de refrescos llega a disminuir, la máquina de refrescos conectada vía inalámbrica contactará al web service de su proveedor local y ordenará exactamente que necesita.
- **Soporte de la industria.** Todos los grandes fabricantes de software/hardware están soportando SOAP y los estándares de web services involucrados. Por ejemplo, la plataforma Microsoft .NET está basada en web services, por lo que ahora es fácil para los componentes escritos en Visual Basic ser instalados como web services, y consumidos por web services usando IBM Visual Age.[116]

El amplio soporte que han tenido los web services por parte de la industria, se debe a que la tecnología de Internet está basada en un sencillo conjunto de estándares de aceptación mundial. Gracias a los inventos de Tim Berners-Lee, cualquiera puede usar un navegador para explorar y disfrutar la vasta cantidad de información almacenada en millones de computadoras alrededor del mundo. Las barreras geográficas tradicionales han caido, y los consumidores tienen la capacidad de comprar productos y servicios de cualquier compañía con un web site.

El Internet está basado en un sencillo y simple conjunto de estándares abiertos. Hypertext Transport Protocol (HTTP), el protocolo de transporte para compartir datos entre máquinas, es rápido de implementar y encabeza el omnipresente protocolo de bajo nivel TCP/IP. Además, Hypertext Markup Language (HTML), el formato para representar datos de forma amigable para el navegador, puede ser entendido por un adolescente en algunas horas. La facilidad de uso ciertamente ayudó la adopción de Internet. *HTML y HTTP hizo posible para las personas compartir información a través de Internet.*[117]

HTTP, ha sido apoyado por satisfacer los nuevos retos de los desarrolladores de aplicaciones distribuidas en Internet. En este escenario, los clientes que se comunican con el servidor sobre Internet enfrentan numerosas barreras potenciales para comunicarse. Los administradores, conscientes de la necesidad de preservar la seguridad en red alrededor del mundo, han implementado ruteadores corporativos y firewalls para no permitir prácticamente todo tipo de comunicación sobre Internet. Usualmente se requiere un milagro para que un administrador de red permita que se abran puertos mas allá de los mínimos permitidos.

Si se tiene la suerte suficiente para que el administrador abra los puertos apropiados para soportar un servicio, las oportunidades que tienen los clientes podrían no ser tan afortunadas. Como resultado, los protocolos propietarios utilizados por DCOM, CORBA y Java RMI no son prácticos para los escenarios de Internet.[118]

En este sentido, **no siempre se acepta ampliamente el que los protocolos propietarios utilicen el puerto 80, reservado para aplicaciones Web**. Como ejemplo de esto se puede citar el esfuerzo que hizo Microsoft para tener un mejor soporte para los escenarios de Internet. Inicialmente esta firma adoptó la estrategia de aumentar sus tecnologías existentes, incluyendo los COM Internet Services (CIS), los cuales permiten establecer una conexión DCIM entre el cliente y el componente remoto sobre el puerto 80. Por varias razones, CIS no fue ampliamente aceptado.[119] De la misma forma, tampoco es bien visto que los protocolos propietarios generen su "HTTP tunneling" por el puerto 80, porque disfrazar un protocolo en otro hace que se piense en soluciones lentas, de programación compleja.

Esta regla limita a arquitecturas como DCOM, CORBA, y Java RMI a trabajar en redes de área local. Hablando de la infraestructura de Microsoft "Distributed Component Object Model" (DCOM), es un soporte para objetos distribuidos que

[115] SHORT, Scott. Op. Cit., p. 3
[116] GLASS. Op. Cit., pp. 22, 23
[117] Ibid., pp. 5,6
[118] SHORT, Scott. Op. Cit., p. 2
[119] Ibid.

permite que una aplicación haga uso de elementos que pertenezcan al Component Object Model (COM) instalados en otro servidor, e incluso esta tecnología ha sido llevada a determinado número de plataformas no basadas en Windows. Pero DCOM no ha ganado amplia aceptación en tales plataformas, por lo que es raramente utilizado para facilitar la comunicación entre computadoras basadas en Windows y las que no lo están. Los vendedores de software para Enterprise Resource Planning usualmente crean componentes para la plataforma Windows el cual les permite comunicarse con el sistema de respaldo solamente a través de un protocolo propietario.[120]

DCOM, CORBA y Java RMI están diseñados para trabajar en un datacenter corporativo, no en Internet. Esto llega a representar una limitante para dichos marcos de trabajo, puesto que algunos servicios entregados para hacer funcionar a una aplicación de e-commerce no podrían residir totalmente dentro del datacenter. Por ejemplo, si la aplicación de e-commerce acepta pagos usando la tarjeta de crédito para bienes y servicios comprados por el cliente, debe obtener los servicios del banco de comercio para procesar la información de la tarjeta de crédito del cliente desde un lugar muy remoto, por ejemplo, un servidor en Internet detrás de decenas de ruteadores o firewalls. Para propósitos prácticos, DCOM y las tecnologías relacionadas -tales como CORBA y Java RMI- son limitadas para aplicaciones y componentes instalados dentro del datacenter corporativo. Dos razones primarias para esto es que por default estas tecnologías descansan en protocolos propietarios y estos protocolos son por herencia orientados a conexión.[121]

Otro punto a favor de los Web Services y en contra de CORBA y DCOM es que la gramática XML SOAP ha ganado aceptación mundial al grado de convertirse un estándar, porque SOAP es un protocolo con información auto descrita –característica ausente en CORBA, DCOM y RMI-. Esto quiere decir que, aún sin una explicación del formato de SOAP, cualquiera que sepa HTML puede comprender que es lo que representa un documento XML SOAP, en contraste con los protocolos CORBA y DCOM, los cuales son binarios, no auto descritos y difíciles de rastrear. Cualquiera que haya escrito programas para CORBA ORB sabe esto.[122]

Al haber efectuado un análisis por separado de las tres arquitecturas para desarrollar sistemas distribuidos, consideramos pertinente realizar un cuadro comparativo que señale las características más importantes de las tres arquitecturas, y de esta manera facilitar la elección de una de ellas para desarrollar el sistema de recepción de órdenes de reparación y reportes de ventas de la industria automotriz:

[120] Ibid., p.1
[121] SHORT, Scott. Op. Cit., p. 1
[122] GLASS. Op. Cit., p. 13

	Arquitectura CORBA	**Arquitectura RMI**	**Arquitectura Web Services**
¿Es una arquitectura orientada a construir objetos distribuidos?	SI	SI	SI
Lenguaje para descripción de interfases	CORBA IDL	Java Interfaces	WSDL
¿Se puede construir un Proxy en el cliente?	SI	SI	SI
La arquitectura permite elaborar objetos simplificados con llamadas a procedimientos remotos (RPC)	NO (la interfaz IDL requiere ser compilada)	NO (la interfaz Java require ser compilada)	SI (el WSDL no requiere compilación adicional)
Servicio para localización de objetos distribuidos	CORBA Naming Service	Java RMI Registry Naming Lookup	UDDI, DISCO
Tipos de Transferencia	RPC Síncrono RPC Síncrono Diferido Unilateral Asíncrona desde la versión 3.0 (en desarrollo)	RPC Síncrono	RPC Síncrono Solicit-Response Unilateral en Cliente Unilateral en servidor Messaging Asincrono
Codificación de los mensajes	Formatos binarios definidos por OMG	Formatos binarios definidos por Sun Microsystems	W3C XML en cualquier dialecto (por ejemplo: SOAP, WSDL, etc)
¿Son los mensajes autodescritos?	NO	NO	SI
Validación de información en los mensajes	NO	NO	XML Schemas (XSD) Document Type Format (DTD)
Velocidad de transferencia de mensajes	Alta	Alta	Mediana
Entidad que recibe y procesa los mensajes	Object Request Broker & Servants (según el lenguaje)	RMI Server & Java Servants	Web Server & CGIs Web Server & Servlets Web Server & Java Server Pages IIS & MSXML4.0 IIS & ASP.NET Stand Alone Daemon Etc
Tipos de mensajes	RPC Por valor RPC Por referencia	RPC Por valor	RPC Por valor RPC Por referencia Orientado a Documento
Protocolo de transporte de mensajes sobre TCP/IP	GIOP IIOP	JRMP JRMP sobre HTTP IIOP	HTTP SMTP
Escenario de Desarrollo	Stand Alone Computer Local Area Network Datacenter Corporativo	Stand Alone Computer Local Area Network Datacenter Corporativo	Stand Alone Computer Local Area Network Datacenter Corporativo Metropolitan Area Network Wide Area Network Internet
Arquitectura amigable con todo tipo de Switch/Router/Firewall	NO	NO	SI
Lenguajes de programación disponibles para desarrollar la arquitectura	C C++ Smalltalk COBOL ADA Java	Java	C C++ Smalltalk COBOL ADA Java Eiffel Modula 3 Perl TCL Objective Python Visual Basic
Arquitectura Basada en Estándares de Aceptación Mundial	NO	NO	SI
Industrias que soportan la Arquitectura	IBM, ORACLE, SUN	SUN	IBM, ORACLE, SUN, Microsoft, PeopleSoft, Otras

Capacidad de programación de sistemas federados	SI	NO	SI
Balanceo de Carga de Trabajo	SI	NO	SI

Tabla 4 Comparación de las arquitecturas CORBA, RMI y Web services

Este cuadro muestra que **los Web services ofrecen comodidad al desarrollador de aplicaciones distribuidas**, con las siguientes *ventajas con respecto a las otras arquitecturas*:

- Mas diversidad en los tipos de mensajes
- Mas diversidad en los tipos de transferencia
- Esquema RPC simplificado
- Auto descripción de los mensajes usando el lenguaje XML y diversos dialectos (SOAP, WSDL, XML Schemas, DTD, etc)
- Validación de mensajes con XML Schemas.
- Más diversidad en los tipos de entidades que reciben y procesan los mensajes
- Protocolo de transporte de mensajes y otros estándares de aceptación mundial
- Más diversidad en los escenarios de desarrollo
- Arquitectura amigable con todo tipo de switch/router/firewall
- Mas diversidad en los lenguajes de programación disponibles para desarrollar la arquitectura
- Número creciente de industrias que soportan la arquitectura
- Escalabilidad de los sistemas desarrollados en esta arquitectura haciendo uso de servicios federados
- Es posible crear una "sociedad" de web services en donde los componentes colaboran para alcanzar sus mentas individuales. Se puede imaginar fácilmente un lugar de mercado en red en donde los Web services se rentan por si mismos al máximo licitador.[123]

La desventaja de los web services es que:

- La velocidad de transferencia de los mensajes del cliente al servidor -y viceversa-, es menor que la de los protocolos binarios.

¿*Por qué son más veloces los protocolos de transporte de CORBA y DCOM que los protocolos basados en texto*? Esto es debido a que CORBA y DCOM usan *una codificación binaria de argumentos y valores de retorno*. Adicionalmente a esto, estos asumen que tanto el remitente como el destinatario tienen completo conocimiento del contexto del mensaje y no codifican ninguna meta-información, tal como los nombres o los tipos de los argumentos. Esta aproximación resulta en un buen desempeño, pero hace difícil para los intermediarios el procesamiento de los mensajes. Y desde que cada sistema usa diferente codificación binaria, es duro trabajo construir sistemas que puedan operar con otras arquitecturas.[124]

Por otra parte, ¿Por qué opera SOAP de manera intrínseca más lento que CORBA, DCOM y RMI? *Debido a que SOAP utiliza XML para codificar mensajes, es muy fácil procesarlos en cada paso del proceso de activación del servicio*. Adicionalmente, la facilidad de corrección de errores en los mensajes de SOAP está dirigida para hacer coincidir rápidamente varias implementaciones SOAP, lo cual es importante porque **la interoperabilidad a gran escala es el principal objetivo de la codificación XML SOAP**.

A primera vista, se observa que un esquema basado en XML podría ser intrínsecamente más lento que un modelo basado en la transferencia binaria de datos, pero no es tan directo como este.[125]

Afortunadamente, esa es la única desventaja de SOAP.

[123] GLASS. Op. Cit., p. 26
[124] Ibid., p. 17
[125] Ibid., p. 17

Los métodos binarios de codificación, tales como los que son utilizados para DCOM y CORBA, son incómodos por ser muy pesados, porque necesitan una compleja infraestructura de soporte para separar al desarrollador de los detalles de implementación. XML es mucho mas ligero en peso y fácil de manejar debido a que puede ser creado y consumido utilizando las técnicas estándar de pre compilación de texto.

Además, una variedad de parsers de XML están disponibles para simplificar más la creación y el consumo de documentos XML en prácticamente toda plataforma moderna. *XML es ligero y es una excelente herramienta de soporte, por tanto, la codificación en XML permite un alcance increíble debido a que prácticamente todo cliente en cualquier plataforma puede comunicarse con el Web service.*[126]

¿Qué otras consideraciones en la operación de SOAP se deben tener en cuenta antes de desarrollar con este método de codificación?

Primeramente, cuando SOAP es utilizado para enviar mensajes a través de Internet, el tiempo de codificar/decodificar los mensajes en cada punto es comparado con el tiempo en que se transfieren los bytes entre puntos finales, por lo que utilizar XML en este caso no es significativo.

Segundo, cuando SOAP es utilizado para enviar mensajes entre dos o mas puntos finales en un entorno cerrado, tal como entre departamentos dentro de la misma compañía, es como si los puntos finales estuvieran corriendo la misma implementación de SOAP. En este caso, existen oportunidades para optimización que son únicas a una implementación en particular. Por ejemplo, un cliente SOAP podría añadir una etiqueta de encabezado HTTP a la petición SOAP que indique que este soporta una optimización particular. Si el servidor SOAP también soporta dicha optimización, podría regresar una etiqueta de encabezado HTTP, en la primera respuesta SOAP que diga al cliente que es aceptable usar esa implementación en comunicaciones subsecuentes. En este punto, el cliente y el servidor podrían iniciar a usar dicha optimización.[127]

Además, XML se apoya en conjuntos de caracteres estándar, lo que resuelve la compatibilidad entre plataformas de hardware. En contraste, los métodos de codificación binarios utilizados por ejemplo en DCOM, CORBA y Java RMI deben tomar en cuenta los asuntos de compatibilidad entre diferentes plataformas de hardware. Por ejemplo, diferentes plataformas de hardware tienen diferentes representaciones binarias internas de números multi-byte. Las plataformas Intel ordenan los bytes de un número multibyte usando la convención “little endian”; por otra parte, muchos procesadores RISC ordenan los bytes de un número multibyte usando la convención “big endian”.

XML evita los problemas de la codificación binaria debido a que este usa un esquema de codificación basado en texto, que descansa en conjuntos de caracteres estándar. Tmbién, algunos protocolos de transporte, tales como SMTP, pueden contener solamente mensajes basados en texto.[128]

Después que hicimos una comparación costo/beneficio, encontramos que la única desventaja es mínima, en comparación a todos los beneficios que ofrece la arquitectura de Web services. *Obviamente, las otras arquitecturas no tienen porqué despreciarse para otro tipo de desarrollos*. Sencillamente se está seleccionando la herramienta con la que se va a trabajar, tal como un carpintero selecciona entre un martillo, una sierra y un desarmador para cortar un trozo de madera.

En los Web services se tienen muchas ventajas pero como es de esperarse, también existen muchas áreas de oportunidad en esta nueva rama del procesamiento de datos en red que se encuentran en desarrollo. Algunos de estos retos suenan intimidantes, hasta que nos damos cuenta que la solución a esos problemas ya existe. Se tienen dos ejemplos de esto, en la sociedad humana y en los organismos biológicos. Ambos exhiben las siguientes propiedades.

- Tolerancia a fallas
- Paralelismo masivo
- Distribución

[126] SHORT, Scott. Op. Cit., p. 7
[127] GLASS. Op. Cit., p. 17
[128] SHORT, Scott. Op. Cit., p.7

- Buena organización
- Auto reparación
- Diseñados en una forma de capas
- Diseñados con componentes simples.

Además de estas características por desarrollar, se perciben ciertos retos técnicos. Estos son:

- **Confiabilidad**. Algunos servidores de Web services serán más confiables que otros. ¿Cómo puede ser esta confiabilidad ser medida y descrita? ¿Qué ocurre cuando un servidor de Web services están fuera de línea temporalmente? En tal caso ¿localizas y usas un servicio alterno provisto por un vendedor diferente, o esperas a que el servicio original regrese? ¿Cómo sabes en que vendedores confiar?
- **Seguridad**. Algunos Web services estarán disponibles públicamente y no asegurados, pero la mayoría de los servicios relacionados con negocios usarán comunicaciones encriptadas con mecanismos de identificación. Al parecer, HTTP sobre Secured Sockets Layer (SSL) proveerá la seguridad básica, pero los servicios individuales necesitarán un mayor nivel de granularidad. ¿Cómo validará a los usuarios un servicio web? ¿Los servicios necesitan poder proveer seguridad en una base de pre-operación? Si se ingresa al servicio con un vendedor que provea los servicios alrededor del mundo, ¿cómo es que estos servicios aprenden acerca de los privilegios de seguridad del usuario?
- **Descubrimiento**. UDDI es el estándar que conduce el descubrimiento dinámico de los web services. Es un estándar joven que probablemente requerirá algunas iteraciones antes de que llegue a convertirse en el buscador de web services de fuerza industrial.
- **Transacciones**. Los sistemas de transacción tradicionales usan la aproximación commit de dos fases, todos los recursos participantes son reunidos y bloqueados hasta que la transacción entera pueda tomar lugar, por la cual son finalmente liberados. Esta aproximación trabaja bien en un entorno cerrado, en donde las transacciones tienen un corto periodo de vida, pero no trabajan bien en un entorno abierto donde las transacciones pueden tomarse horas o aún días. Microsoft soporta un esquema alternativo, llamado transacciones de compensación, en su nuevo sistema XLANG para procesos de negocios distribuidos. ¿Deben esta clase de transacciones ser integradas en web services? En tal caso, ¿Cuál es el traslape entre esta aproximación y los estándares propuestos como XAML, un lenguaje XML de etiquetas para soportar transacciones tradicionales?[129]

Otros retos técnicos retos técnicos a alcanzar por los Web services son:

- **Escalabilidad**. Desde que es posible exponer sistemas de componentes existentes como Enterprise JavaBeans (EJB) como web services, debe ser posible apoyarse en el balanceo de cargas y otros mecanismos de escalabilidad que ya existen. ¿O es que existen bloques aun no vistos que obstaculizan este camino? ¿Se tiene ya la necesidad de un nuevo tipo de servidor de aplicaciones de "web services"?
- **Manejabilidad.** ¿Qué tipo de mecanismos son requeridos para maneja un sistema de alta distribución? Debido a que las propiedades de un sistema son función de las propiedades de sus partes, ¿los administradores de diversos Web services necesitan coordinarse en una forma en particular? ¿Y es posible asignar a un tercero la administración de los web services u otros web services? ¿Cómo cobrarás por el uso de los Web services? ¿Será usando el modelo dominante basado en suscripción, o pago según el uso?
- **Pruebas.** Cuando un sistema está compuesto de varios web services cuya localidad y características son potencialmente dinámicas, las pruebas y la corrección de errores toman una entera nueva dimensión. ¿Cómo alcanzarás los tiempos de respuesta predecibles? ¿Cómo eliminarás errores de programación en los web services que vienen de diferentes vendedores, almacenados en diferentes ambientes y sistemas operativos?[130]

Por todas las ventajas expuestas en esta sección y a pesar de los ligeros inconvenientes y retos téncnicos, se ha seleccionado a los Web services como plataforma arquitectónica de software para desarrollar la aplicación del uso de la industria automotriz.

[129] Ibid., p. 25
[130] Ibid.

2.6 Método de validación: Autómatas Finitos Deterministicos

¿Qué son las expresiones regulares? Este concepto tiene su fundamento en el álgebra de conjuntos y las matemáticas vectoriales, en la rama de los lenguajes formales y autómatas. Dicho contexto define a las expresiones regulares como la notación que permite definir de manera precisa un lenguaje[131].

Técnicamente hablando, las expresiones regulares definen a los lenguajes tipo 3 en la clasificación de Chomsky con respecto a las restricciones en las producciones de una gramática[132]. Las expresiones regulares básicas se definen a través de tres reglas:

1. epsilon es una expresión regular designada por {epsilon}, es decir, el conjunto que contiene la cadena vacía.
2. Si a es un símbolo de Sigma, a es una expresión regular designada por {a}. a puede ser una cadena o una letra en el alfabeto
3. Suponiendo que r y s son expresiones regulares representadas por los lenguajes L(r) y L(s), entonces si se tiene la operación:
   1) r|s, es entonces expresión regular resultado de L(r) U L(s). Es decir, la unión de los conjuntos tipo lenguaje permite que se utilizen cuerdas del lenguaje L(r) y L(s). Esta es la operación OR.
   2) (r)(s) es el producto de los lenguajes L(r)L(s). Esta es la operación concatenación.
   3) r* representa L(r)*. Esta es la operación cerradura de Kleene.

La procedencia de los operadores es la siguiente:

I. Cerradura *
II. Concatenación (r)(s)
III. OR |

Por definición,
a*={ epsilon a aa aaa aaaa …}
b*a={a ba bba bbba bbbba…}

Las expresiones regulares cuentan con ciertas propiedades algebráicas. Sean r, s y t expresiones regulares. Entonces:

1. r|s=s|r (*Conmutatividad de union*)
2. (r|s| epsilon)= (r| epsilon |s)= (epsilon |r|s)=… (*Conmutatividad de union con permutación múltiple*)
3. r|(s|t)= (r|s)|t (*Asociatividad de union*)
4. (rs)t=r(st) (*Asociatividad con concatenación*)
5. r(s|t)=rs|rt (*Distributividad con concatenación de union*)
6. r*=(r| epsilon)* (*Cerradura*)
7. r**=r* (*Idempotencia*)
8. r+=rr*=r*r (*Elemento idéntico*)

Dicho mas claramente, una expresión regular es **el patrón que puede seguir un conjunto de datos**, sea este finito o infinito. A continuación se citan ejemplos de expresiones regulares:

Expresión Regular	Ejemplo(s) de cadenas válidas
`Capitulo \d`	Capitulo 1
`Capitulo \d`	Capitulo 1
`a*b`	b ab aab aaab
`[xyz]b`	xb yb zb
`a?b`	b ab
`a+b`	ab aab aaab

[131]Un lenguaje es el conjunto de cuerdas que se pueden formar en un alfabeto, así como un conjunto de reglas a seguir con ese alfabeto.
[132]Las gramáticas son los conjuntos G = {N Sigma P S}, en donde N es el conjunto de elementos no terminales, Sigma es el alfabeto de la gramática, P es el conjunto de producciones o reglas de derivación y S es el símbolo inicial de la gramática y en este caso, S pertenece al conjunto de los elementos no terminales.

`[a-c]x`	ax bx cx
`[-ac]x`	-x ax cx
`[ac-]x`	ax cx –x
`[^0-9]x`	Algún carácter no digito seguido por x
`\Dx`	Algún carácter no digito seguido por x
`Capitulo\s\d`	Capitulo 0, Capitulo 1,…, Capitulo 9
`(ma){2}`	mama
`(pa\s){2}`	pa pa
`.abc`	Un carácter seguido por la cadena abc
`(a|b)+x`	ax bx aax bbx abx bax …
`a{1,3}x`	ax aax aaax
`a{2,}x`	aax aaax aaaax aaaaax …
`\w\s\w`	**character character** (la palabra **character** es una palabra reservada, por lo que se simboliza con un **\w**)
`[a-zA-Z-[OI]]*`	Una cadena compuesta de letras mayusculas y minúsculas, exceptuando la “O” y la “I”
`\.`	El punto (sin la diagonal invertida un punto en una expresión regular simboliza cualquier carácter)
`\n`	Linefeed
`\r`	Carriage return
`\t`	Tabulador
`\\`	Diagonal invertida
`|`	Barra vertical
`\-`	Guión
`\^`	Acento circunflejo
`\?`	Signo de interrogación de cierre
`\*`	Asterisco
`\+`	Signo mas
`\{`	Llave de apertura
`\}`	Llave de cierre
`\(`	Paréntesis de apertura
`\)`	Paréntesis de cierre
`\[`	Corchete de apertura
`\]`	Corchete de cierre
`\p{L}`	Una letrade algún lenguaje
`\p{Lu}`	Una letra mayúscula, de algún lenguaje
`\p{Ll}`	Una letra minúscula, de algún lenguaje
`\p{N}`	Un número: romano, fracciones, etc
`\p{P}`	Un signo de puntuación
`\p{Sc}`	Un signo de moneda de algún lenguaje

Tabla 5 Ejemplos de expresiones regulares

Las expresiones regulares son versátiles en el sentido de que limitan el contenido de una cadena. Por ejemplo, si se desea limitar el contenido de una cadena para que su valor se encuentre dentro del rango de 0 a 255, se debe usar la siguiente expresión regular:

```
[1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]
0 a 99      |100 a 199  |200 a 249  |250 a 255
```

Esta expresión regular podría ser útil para describir una direccion IP, por ejemplo.

Se ha hablado un poco de las expresiones regulares pero, **¿qué aplicación tienen en el área de programación?** Las expresiones regulares sirven fundamentalmente para *construir analizadores léxicos*, conocidos en la disciplina de los compiladores como "scanners". Un scanner toma un programa fuente y lo transforma en tokens.

¿Cómo está construido un scanner? Un scanner utiliza un autómata finito determinístico para buscar expresiones regulares en una cadena. De esta forma, puede transformar una cadena completa en "tokens".

Ahora bien,¿Qué son los Autómatas finitos? Un autómata finito está formado por los siguientes elementos:

AF = {Sigma Q q_0 delta F}, en donde Sigma es el alfabeto del autómata finito, Q es un conjunto de estados, q_0 es el estado inicial, delta es un conjunto de transiciones y F es el conjunto de estados finales.

Existen dos tipos de Autómatas finitos:

- Autómatas Finitos Determinísticos (AFD). Este tipo de autómatas están compuestos por un solo estado inicial. Además, de un nodo sale una rama usando una etiqueta única, mejor conocida como "clase de símbolos".
- Autómatas Finitos No Determinísticos (AFN). Este tipo de autómatas tiene varios estados iniciales. En este caso, del mismo nodo salen dos o más ramas usando la misma etiqueta o "clase de símbolos.

Algo fundamental en los Autómatas Finitos Determinísticos es que necesitan una expresión regular por cada clase que se quiera reconocer o "escanear". Si se tiene más de una clase, se deben unir todas ellas con el operador "OR" definido para las expresiones regulares.

¿Cómo se obtiene el AFD a partir de una expresión regular? Se realiza a través de cuatro pasos:

1. Obtener el sistema de transición.
2. Obtener el diagrama de transición (AFN)
3. Obtener el Autómata Finito Determinístico (su diagrama con tabla)
4. Obtener el AFD mínimo, cuyo sistema de transición cuenta con 1 estado inicial y 1 o más estados finales.

¿De qué forma se puede programar un AFD para una expresión regular, por ejemplo, de un correo electrónico? El primer paso es obtener la expresión regular a validar. Esta expresión regular puede ser:

$$c^{+}@(c^{+}.)^{+}c^{+}$$

En donde c es un símbolo del conjunto {$ + - _ A-Z a-z 0-9}, . es el símbolo del conjunto {.} y @ es el símbolo del conjunto {@}.

El segundo paso es construir un diagrama de estados utilizando la información de la expresión regular:

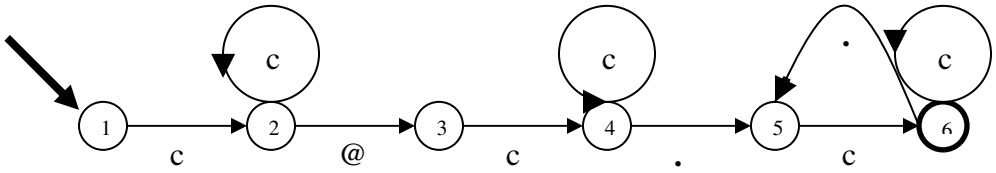


Ilustración 30 Diagrama de estados obtenido con una expresión regular

El tercer paso es construir un arreglo con la información del autómata (estados, clases de símbolos, estado inicial, estado final:

	c	**@**	**.**	**¿Es Estado Final?**
Estado 1 (inicial)	2	Error	Error	No
Estado 2	2	3	Error	No
Estado 3	4	Error	Error	No
Estado 4	4	Error	5	No
Estado 5	6	Error	Error	No
Estado 6	6	Error	5	Si

Tabla 6 Arreglo obtenido a partir del diagrama de estados

Finalmente, se deben programar tres rutinas: *una para construir el arreglo, otra para saber qué clase de símbolo es un carácter, y otra para validar la cadena dando seguimiento al arreglo*:

A continuación se mostrarán dichas rutinas en el lenguaje de programación Visual Basic for Applications de Microsoft Access, el cual muestra las propiedades de los autómatas finitos determinísticos:

Declaraciones

```
Dim AFD(1 To 6, 1 To 4) As Integer
Dim columnaEstadoFinal As Integer
```

Construye Autómata Finito Determinístico

```
Sub construyeAFD()
'AFD de la expresión regular c+@(c+.)+c+
'En donde c es un simbolo de entre $ + - _ A-Z a-Z 0-9
'         c+ es uno o más simbolos descritos anteiormente
'         . es un punto
'         @ es una arroba
'El AFD tiene la siguiente forma:
' j 1 2 3 estado final
'i********************
'1* 2 x x false
'2* 2 3 x false
'3* 4 x x false
'4* 4 x 5 false
'5* 6 x x false
'6* 6 x 5 true
'i es renglón, j es columna, x es -1 o simbolo inválido
columnaEstadoFinal = 4
For i = 1 To 6
    For j = 1 To 3
    AFD(i, j) = -1
    Next j
    AFD(i, columnaEstadoFinal) = False
Next i
'escribir el AFD
AFD(1, 1) = 2
AFD(2, 1) = 2
AFD(2, 2) = 3
AFD(3, 1) = 4
AFD(4, 1) = 4
AFD(4, 3) = 5
AFD(5, 1) = 6
AFD(6, 1) = 6
AFD(6, 3) = 5
AFD(6, columnaEstadoFinal) = True
End Sub
```

Valida la cadena a partir del AFD y de la rutina Clase Simbolo

```
Function validaEmailAddress(cadena) As Boolean
Dim simbolo As String, apuntador As Integer
Dim estado As Integer, clase As Integer

If IsNull(cadena) Then cadena = ""
cadena = Trim(cadena)
longitud = Len(cadena)
estado = 1
apuntador = 1
If longitud > 1 Then
```

	construyeAFD
	Do
	simbolo = Mid(cadena, apuntador, 1)
	clase = ClaseSimbolo(simbolo)
	If estado = -1 Or clase = -1 Then
	validaEmailAddress = False
	Exit Function
	Else
	estado = AFD(estado, clase)
	apuntador = apuntador + 1
	End If
	Loop While apuntador <= longitud
	If estado = -1 Then
	validaEmailAddress = False
	Exit Function
	ElseIf AFD(estado, columnaEstadoFinal) Then
	validaEmailAddress = True
	Else
	validaEmailAddress = False
	End If
	Else
	validaEmailAddress = False
	End If
	
	End Function

Clasifica el carácter de entrada y le asigna	Function ClaseSimbolo(caracter As String) As Integer Select Case Asc(caracter) Case 36, 43, 45, 65 To 90, 95, 97 To 122, 48 To 57 '$ + - _ A-Za-Z0-9 ClaseSimbolo = 1 Case 64 '@ ClaseSimbolo = 2 Case 46 '. ClaseSimbolo = 3 Case Else ClaseSimbolo = -1 End Select End Function

Utiliza la subrutina de validación	Sub main()
	Dim SaulMorenoMotte As String
	For i = 1 To 5
	SaulMorenoMotte = InputBox("Ingrese la cadena a Validar", _
	"Validación con AFD", "vigmms@hotmail.com")
	If validaEmailAddress(SaulMorenoMotte) Then
	MsgBox SaulMorenoMotte + " si es una dirección de correo electrónico", _
	vbOKOnly + vbInformation, "Validación con AFD"
	Else
	MsgBox SaulMorenoMotte + " no es una dirección de correo electrónico", _
	vbOKOnly + vbCritical, "Validación con AFD"
	End If
	Next i
	End Sub

Tabla 7 Rutinas para validar un tipo de datos a partir de una expresión regular

A continuación, se muestra *una corrida del programa para verificar que funciona adecuadamente*:

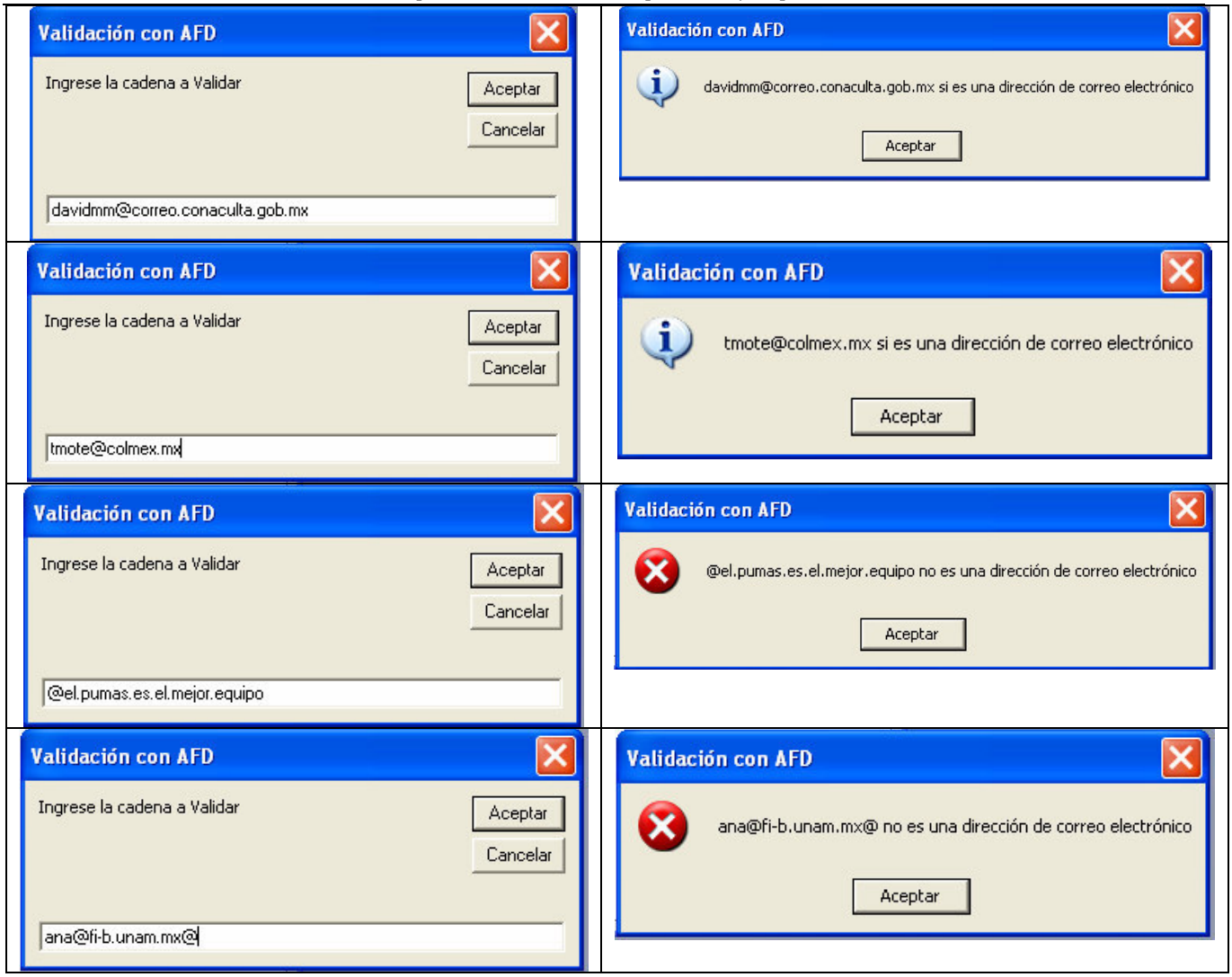


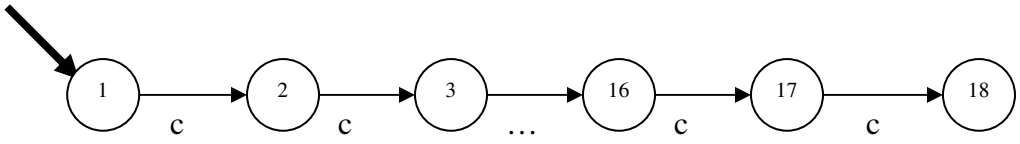
Tabla 8 Corrida del programa de validación de datos

¿Cómo podría diseñarse un AFD para validar una expresión regular de un Número de Serie de un Vehículo, o VIN? De nueva cuenta, el primer paso es obtener la expresión regular a validar. Esta expresión regular puede ser:

```
c{17}
```

En donde c es un símbolo del conjunto {A-Z a-z 0-9}.

El segundo paso es construir un diagrama de estados utilizando la información de la expresión regular:



El tercer paso es construir un arreglo con la información del autómata (estados, clases de símbolos, estado inicial, estado final:

	c	**¿Es Estado Final?**
Estado 1 (inicial)	2	No
Estado 2	3	No
Estado 3	4	No
…	…	…
Estado 17	18	No
Estado 18	Error	Si

Nuevamente, se deben programar tres rutinas: una para construir el arreglo, otra para saber qué clase de símbolo es un carácter, y otra para validar la cadena dando seguimiento al arreglo[133].

En resumen, ¿Qué debe hacerse para validar una cadena a partir de una expresión regular? Consideramos los siguientes elementos:

1) Creamos una **expresión regular**.
2) Construimos un **diagrama de estados** a partir de una expresión regular. Revisamos que ésa grafica tuviera un solo estado inicial.
3) Construimos **un arreglo** a partir del diagrama de estado, y definimos clases de símbolos y estados finales.
4) Programamos **un ciclo** capaz de:
   a. *Revisar* cualquier cadena, símbolo por símbolo.
   b. *Recorrer* la cadena al mismo tiempo que se recorre el arreglo y se verifica la valides o invalidez del estado.
   c. *Dictaminar* que la cadena es invalida si se encuentra una clase de símbolos no permitida.
   d. *Dictaminar* que la cadena es inválida si después de recorrer la cadena, en el arreglo se llego a un estado inválido.
   e. *Dictaminar* que la cadena es valida si después de recorrer la cadena en el arreglo se llego a un estado valido.

¿Por qué se ha señalado el funcionamiento de los AFD? ¿Qué objeto tuvo haber ejemplificado su utilización con la expresión regular de un correo electrónico? ¿Qué tienen que ver los AFD con los Java Web Services? *Hemos* considerado ésta metodología para *ilustrar el gran trabajo que está detrás de validar cada uno de los tipos de datos*, y también para señalar que *los XML Schemas proveen la magnífica ventaja de que construyen automáticamente los Autómatas Finitos Determinísticos,* **tan solo pasandoles como parámetro la expresión regular a validar**. En la **Sección 3.2.1** se mostrará como hacer esta tarea usando una etiqueta de XML llamada pattern cuyo atributo value adquirirá el valor de la expresión regular, por ejemplo `[A-Z]{3}-\d{3}`.

Los XML Schemas no solo tienen la ventaja de que construyen automáticamente sus AFD. *También descubrimos que para validar una nueva expresión regular, o hacer correcciones a una existente, no se requiere recompilar código alguno por parte de un programador*, sino sencllamente *editar el archivo XSD en donde se encuentre almacenado el XML Schema*. Además, los XML Schemas tienen un poderoso conjunto de tipos de datos definidos, por lo que no se requiere escribir una expresión regular para definir tipos de datos como fecha, por ejemplo, en donde se tienen un interesante conjunto de reglas, entre ellas:

1. Elementos declarados para ser del tipo fecha deben seguir la forma CCYY-MM-DD
2. El rango de CC es 00-99
3. El rango de YY es 00-99
4. El rango de MM es 01-12

[133] Para más detalles sobre Compiladores, Vid. KENNETH C. Louden. Compiler Construction Principles and Practice. cfr., TERRANCE W Pratt & Marvin V. Zelkowitz. Lenguajes de Programación, Diseño e Implementación. Ed. Prentice Hall
Para más detalles sobre Lenguajes Formales y Autómatas Vid. Jean Paul TREMBLAY. Paul G. SORENSON. The Theory And Practice of Compiler Writing. Ed. McGrawHill International Editions. Computer Science Series.

5. El rango DD está definido de la siguente forma:
    a. 01-28 si MM es 02
    b. 01-29 si MM es 02 y YY es un año bisiesto
    c. 01-30 si MM es 04, 06, 09 u 11
    d. 01-31 si MM es 01, 03, 05, 07, 08, 10 o 12

A pesar que en Internet todavía existen pocas referencias de cómo validar instancias de XML SOAP Requests a partir de un XML Schemas, en el presente trabajo de investigación mostraremos como hacer esta tarea en la **Sección 4.4**.

3 Fundamentos de Web Services

3.1 Protocolo HTTP

El protocolo de transferencia de hipertexto HTTP es un protocolo de nivel de aplicación con la ligereza y velocidad necesaria para sistemas de información *distribuidos*, *colaborativos*, basados en diversos medios y recursos. Es un protocolo *genérico*, *sin estado, orientado a objetos*, el cual puede ser utilizado para varias tareas, tales como name servers y sistemas de administración de objetos distribuidos, a través de la extensión de sus métodos request (conocidos también como comandos HTTP). Una característica de HTTP es el tipo y la negociación de representación de datos, permitiendo que los sistemas sean construidos independientemente de los datos que se están transfiriendo[134].

En la definición anterior se menciona que HTTP es un protocolo sin estado. ¿Qué se debe entender en esta frase? Esto significa que dondequiera que el cliente haga una solicitud de información, la conexión al servidor es cerrada después que el cliente recibe la respuesta. Por tanto, si algún estado debe mantenerse entre el cliente y el servidor, el servidor debe pasar información sobre el estado con la respuesta al cliente. Esto permitirá al servidor recuperar esta información del cliente cuando este recibe la siguiente requisición. Por ejemplo, si se crea un Web site que muestra el contenido del usuario específicamente, se desearía crear un mecanismo que retenga la información acerca del usuario actual para demostrar su contenido personalizado[135].

Aquí se mencionó una aplicación de HTTP, pero, en términos generales, ¿para qué sirve HTTP? HTTP es el protocolo estándar propiedad del consorcio para la World Wide Web (W3C) utilizado para transferir documentos en Internet. **HTTP es utilizado para diversas tareas, adicionalmente a su uso original de transferencia de hipertexto**. Por ejemplo, los Web services basados en XML usan HTTP para hacer sus comunicaciones. Es un protocolo genérico[136].

¿Qué ventajas obtiene quien construye sus aplicaciones sobre el protocolo HTTP? Para responder, se pueden listar todas sus propiedades y recursos[137]:

- **Es amigable con el firewall**. Los protocolos anteriores tales como Distributed Component Object Model (DCOM) no lo son. La mayoría de los firewalls tienen el abierto al menos el puerto 80 para el tráfico de HTTP.
- **Tiene una infraestructura de soporte robusta.** Muchas tecnologías han sido introducidas en el esfuerzo de incrementar la escalabilidad y disponibilidad de las aplicaciones basadas en http.
- **Es inherentemente libre de estado**. La naturaleza en la carencia de estado de http ayuda a asegurar que las comunicaciones entre el cliente y el servidor son confiables, especialmente a través de Internet. Las caídas intermitentes de las conexiones presentan problemas para los protocolos como DCOM y CORBA.
- **Es simple.** El protocolo http está compuesto de una sección header y una sección body.
- **Mapea perfectamente los intercambios de mensajes de tipo RPC**. HTTP es un protocolo natural para las comunicaciones de tipo RPC debido a que una petición es siempre acompañada de una respuesta.
- **Es abierto**. Prácticamente cada sistema basado en redes soporta http.

¿Qué problema resuelve HTTP de mejor manera que otros protocolos? Uno de los temas centrales de los sistemas de objetos distribuidos es determinar como el cliente y el servidor –u otro actor- pueden comunicarse uno con otro. Los equipos de cómputo no siempre residen en la misma LAN, ya que el cliente podría potencialmente comunicarse con el servidor sobre Internet. Por tanto, **el protocolo de transporte debe ser diseñado igualmente tanto para ambientes de LAN como para la Internet**.

---

[134] Vid. http://www.serverwatch.com/tutorials/article.php?1354871
[135] MICROSOFT CORPORATION. "The Underlying Technologies of XML Web services" En: Developing XML Web Services Using Microsoft© ASP.NET. p.3
[136] Ibid., p.2
[137] SHORT, Scott. Op. Cit., pp. 54,55

En secciones anteriores de este trabajo se marcó el hecho de que tecnologías tales como DCOM, CORBA y Java RMI están pobremente diseñadas para soportar comunicaciones entre el cliente y el servidor sobre Internet. Los protocolos tales como Hypertext Transfer Protocol (HTTP) y Simple Mail Transfer Protocol (SMTP) son protocolos de Internet probados. HTTP define un patrón de mensajería request/response para enviar una petición y obtener su respuesta asociada. SMTP define un protocolo de mensajes de ruta programable para comunicación asíncrona[138].

Además, el que HTTP sea un protocolo que no depende del estado de la aplicación debe verse como otra ventaja. La razón es que éstas no dependen de una conexión continua entre el cliente y el servidor. Eso hace que HTTP sea un protocolo ideal para configuraciones de alta disponibilidad como los firewalls. Si el servidor que ha manejado la requisición original del cliente llega a no estar disponible, las requisiciones subsecuentes pueden ser de manera automática dirigidas a otro servidor sin que el cliente tenga conocimiento o tenga que realizar alguna acción adicional[139].

HTTP no solo rompe barreras técnicas. De hecho ¡también rompe barreras humanas! Los administradores de red prefieren HTTP por ser un servicio popular y de fácil administración. Los empleados han venido confiando tanto en e-mail como en los Web browsers, y los administradores de red tienen aplicaciones con un alto nivel de comodidad para soportar tales servicios. Las tecnologías tales como Network Address Traslation (NAT) y servidores Proxy proveen una vía de ingresar a Internet vía HTTP dentro de otra LAN corporativa aislada. Los administradores casi siempre expondrán un servidor SMTP que resida dentro del firewall. Los mensajes enviados a este servidor entonces tendrán que ser ruteados a su destino final vía Internet.

Si se piensa en una aplicación crítica como el software de procesamiento de información de las tarjetas de crédito, donde se requiere una respuesta inmediata del banco de comercio para determinar si la orden debe ser enviada al sistema ERP o no, quien se ajusta bien a esta tarea es HTTP, con su patrón de petición/respuesta[140].

HTTP maneja URLs, que son casos particulares de los URIs. ¿Qué semejanzas y diferencias tienen entre sí? La dirección de un recurso está especificada en HTTP a través de un mecanismo conocido como Uniform Resource Locator (URL). Hablando estrictamente, el mecanismo que es utilizado en HTTP es un Uniform Resource Identifier, pero podemos también pensar en el como un URL.

Los URIs son claramente un esquema más general para localizar recursos en Internet, que están enfocados más sobre el recurso y menos en la dirección. En teoría, un URI pudría buscar la copia más cercana de un documento duplicado en alguna parte, o localizar el documento que fue movido de un lugar a otro. Los URLs son el conjunto de esquemas de URI que han nombrado el recurso y contienen instrucciones explícitas sobre cómo utilizar el recurso[141].

La estructura de un URL es bien conocida por su uso cotidiano. Su sintaxis es como sigue[142]:

```
http://host[:port][path[?querystring]]
```

El siguiente es un ejemplo de un URL:
`http://www.woodgrovebank.com/accts.asp?AccNo=23`
En el ejemplo anterior, `www.woodgrovebank.com` es el host, `accts.asp` es el path y `AccNo=23` es el query string. Si el número de puerto no está especificado, como en el ejemplo anterior, es utilizado el puerto por default para HTTP, el cual es el 80.

En los estándares del protocolo HTTP, además de definirse cómo representar la localización del recurso, también se definen las estructuras generales del esquema Solicitud/Respuesta. Su sintaxis es como sigue[143]:

---

[138] Ibid., p.5
[139] Ibid., p.6
[140] Ibid.
[141] MICROSOFT CORPORATION. The Underlying Technologies..., p.3
[142] Ibid.

```
[METHOD] [REQUEST URI] HTTP/[VER]
[FIELDNAME1] [FIELDVALUE1]
[FIELDNAME2] [FIELDVALUE2]
[BLANK LINE]
[REQUEST_BODY_IF_ANY]

HTTP/[VER]
[FIELD1] [VALUE1]
[FIELD2] [VALUE2]
[BLANK LINE]
[DOCUMENT_CONTENT_HERE]
```

En el caso de un Web Service, tanto en el cuerpo de la solicitud ó "request body" como en la respuesta a la solicitud ó "document content" viajan mensajes codificados en XML

A pesar que la notación anterior está bastante condensada, se dificulta un poco su lectura. A continuación se mostrará la sintaxis de una solicitud y una respuesta por separado[144]:

Estructura de una solicitud http	Estructura de una respuesta HTTP
method URL Version headers a blank line message body	Version Status-Code Description headers a blank line message body

Tabla 9 Solicitudes y respuestas de información en el protocolo HTTP

La diferencia más notoria entre estos dos tipos de mensajes, es que la solicitud HTTP menciona un método, mientras que la respuesta no. ¿Qué tipos de métodos son soportados por HTTP? La primer línea en una petición HTTP es conocida como la línea de request, y los métodos que puede soportar son los descritos a continuación[145]:

- **OPTIONS**
- **GET**
- **HEAD**
- **POST**
- **DELETE**
- **TRACE**
- **CONNECT**
- **Extensión-method**

[143] Vid. http://www.garshol.priv.no/download/text/http-tut.html
[144] Microsoft Corporation. The Underlying Technologies..., pp.4,5
[145] Ibid., p. 4

Para ejemplificar el uso de la sintaxis y los métodos de http, el siguiente cuadro expone *un ejemplo de una solicitud HTTP y su respuesta correspondiente*[146]:

Ejemplo de una solicitud HTTP	POST /TheStockExchange/Trading/GetStockPrice.asp http/1.1 Host: localhost Content-Type: application/x-www-form-urlencoded Content-Length:11 <?xml version="1.0" encoding="utf-8"?> <Symbol=MSFT />
Ejemplo de la respuesta HTTP correspondiente	HTTP/1.1 200 OK Content-Type: text/xml; charset=utf-8 Content-Length:75 <?xml version="1.0" encoding="utf-8"?> <stock symbol="MSFT" Price="71.50" />

Tabla 10 Ejemplo de solicitud y respuesta en el protocolo HTTP

De entre todos los métodos de HTTP mencionados, ¿cuáles son utilizados para comunicarse con un servicio Web? Los métodos GET y POST para solicitudes son ideales para comunicarse con un XML Web service. Estos métodos están diseñados específicamente para enviar datos al servidor Web y obtener uno de sus recursos específicamente. Esto hace posible colocar un modelo de capas de llamada de función arriba de estos métodos, lo cual es exactamente lo que el modelo de servicios Web basados en XML necesita[147].

De estos métodos, ¿cuál es el método más adecuado para enviar datos al servidor? En primer lugar, POST debe ser utilizado cuando la petición causa un cambio permanente en el estado del servidor (tal como añadir un elemento en una lista de datos) y GET cuando este no es el caso (por ejemplo, al momento de efectuar una búsqueda) Si los datos pueden ser grandes (más de 256 caracteres) es un poco peligroso usar GET porque el URL puede ser recortado en el tránsito del mensaje. Algunos sistemas operativos no permiten variables de ambiente mayores a 256 caracteres, por lo que la variable de ambiente que almacena la parte "?---" de la petición puede ser **silenciosamente truncado**. **Este problema se evita con el método POST**, en cuyo caso **los datos son enviados en el cuerpo del mensaje**, y los mismos alcanzan su objetivo sin este problema[148].

Considere la siguiente solicitud HTTP-GET

```
GET /Trading/GetStockPrice.asp?Symbol=MSFT HTTP/1.1
Host: localhost
```

La característica más importante de la línea de solicitud es el querystring. El querystring es la porción del URI que sigue al signo de interrogación, y consiste de un conjunto de pares nombre/valor codificados en el URL.

En una solicitud HTTP-GET, comúnmente no hay cuerpo del mensaje. La respuesta para una solicitud http-GET es tan solo una respuesta http estándar.

Ahora, considere la siguiente solicitud HTTP-POST

[146] Ibid., p. 4, 5
[147] Ibid., p. 6
[148] Vid. http://www.garshol.priv.no/download/text/http-tut.html

```
POST /Trading/GetStockPrice.asp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

Symbol=MSFT
```

En el código anterior, nótese que no hay querystring como parte del URI. Esto es porque la información acerca de la solicitud está contenida en el cuerpo del mensaje. Esta característica de la solicitud *HTTP-POST hace una forma muy conveniente de pasar grandes conjuntos de datos al servidor en contraste con un HTTP-GET* en donde el tamaño del querystring está restringido a 1024 bytes. También, transmitir los datos como parte del cuerpo del mensaje impone menos restricciones en el tipo de datos que están siendo enviados al servidor[149].

Se ha observado que estructuralmente, el protocolo HTTP es ideal para transmitir mensajes de Web services, dadas sus características de especificación. **La asociación de estos protocolos –HTTP y XML– se le conoce como "Protocol Binding"**. ¿Qué definición formal se puede dar de esta relación? La forma en que el mensaje SOAP es transportado por un protocolo de transporte en particular es conocido como protocol binding. Un protocol binding puede ser definido para explotar algunas características únicas del protocolo de transporte. HTTP Post binding extiende el protocolo por lo que los firewalls dedicados a la administración de conexiones HTTP tienen la posibilidad de filtrar, y dejar pasar los mensajes SOAP[150].

Entonces, el protocolo de transporte asociado a SOAP por especificación es http. La especificación SOAP describe únicamente un protocol binding: enviar mensajes SOAP a través de http POST.

La mayoría de las implementaciones SOAP, incluyendo .NET, soportan el protocolo HTTP. Debido a que la mayoría de los sistemas soportan HTTP, ha llegado a ser la opción del protocolo por excelencia para asegurar que un Web service tiene un alto grado de interoperabilidad entre diferentes plataformas[151].

HTTP define un encabezado especial para utilizarse al momento de transportar un xml hacia un servicio, etiquetado como SOAP Action. El encabezado SOAPAction es utilizado para comunicar el propósito del mensaje SOAP. El URI puede ser representado en cualquier formato y no es necesario que lo deba resolver algún servidor de DNS[152].

El valor del encabezado SOAPAction puede quedar en blanco si el propósito del mensaje SOAP está contenido dentro del header del HTTPRequest. El HTTP Request es el primer registro en el header y contiene la acción (en este caso, siempre POST) y el URI objetivo. Si el URI en el encabezado http request comunica adecuadamente el propósito del mensaje SOAP, lo siguiente puede ser válido:

```
SOAPAction: ""
SOAPAction:
```

Si la conexión punto a punto HTTP fuera un "tubo transparente" ¿Qué datos se verían pasar por ella? Para concluir esta sección, se mostrarán los ejemplos de cómo se ve un XML que se envía a un servicio, y un XML de respuesta, todo transportado en HTTP[153]:

[149] Microsoft Corporation. The Underlying Technologies, pp. 6,7
[150] SHORT, Scott. Op. Cit., p.54
[151] Ibid.
[152] Ibid., pp. 55, 56
[153] Ibid.

<table>
<tr><td>SOAP Request utilizando el HTTP como su protocolo asociado</td><td>Una petición http está compuesta de dos partes, un encabezado y un cuerpo. El encabezado contiene información acerca de la petición y acerca de l cliente que envió la petición. El cuerpo sigue al headery está delimitado por dos pares de “carriage-return/linefeed”. El cuerpo contiene la carga de datos, la cual en este caso podría ser el mensaje SOAP. Aquí se muestra un ejemplo de la petición http que contiene un mensaje SOAP.

<pre>POST /SomeWebService HTTP/1.1
Content-Type: text/xml
SOAPAction: "http://somedomain.com/SomeWebService.wsdl"
Content-Length: 243
Host: sshort3

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/">
  <soap:Body>
    <Add>
      <x>2</x>
      <y>2</y>
    </Add>
  </soap:Body>
</soap:Envelope></pre>
El encabezado HTTP para el mensaje SOAP es similar a la petición HTML, con un par de diferencias: El encabezado Content-Type es siempre establecido en text/xml, y el cuerpo del mensaje contiene un mensaje SOAP. La otra diferencia es que cada petición SOAP http POST debe contener un encabezado SOAPAction.</td></tr>
<tr><td>SOAPResponse usando HTTP como protocolo de transporte</td><td>La respuesta HTTP es utilizada para comunicar los resultados del SOAP request:

<pre>HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sun, 25 Mar 2001 19:44:55 GMT
Content-Type: text/xml
Content-Length: 243

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResponse>
      <result>4</result>
    </AddResponse>
  </soap:Body>
</soap:Envelope></pre>
Una vez más, el tipo MIME es establecido en text/xml. Para los mensajes RPC, el cuerpo http contiene el mensaje de respuesta SOAP. De otra manera, el cuerpo de http podría quedar vacío. Debido a que el mensaje de petición de ejemplo resulta en el mensaje de respuesta generado en el servidor, el http body contiene los resultados.</td></tr>
</table>

Tabla 11 Ejemplo de solicitud y respuesta de XML SOAP en el protocolo HTTP

Este ejemplo ilustra el concepto de Web services de forma completa: se envía al servidor un par de parámetros en un sobre de XML, -x e y- transportado en HTTP, cuyo SOAPAction es:

`"http://somedomain.com/SomeWebService.wsdl"`

el servicio extrae x e y del sobre, hace la suma y devuelve el resultado, también en formato XML. Aquí se observa prematuramente que dicho mensaje está codificado en SOAP –que es una gramática en particular de XML-. La siguiente sección hablará de porqué SOAP es una buena opción para codificar los mensajes, y representar parámetros como x e y.

3.2 Elementos Esenciales del lenguaje XML

El lenguaje XML es un lenguaje para estructurar datos. *Los datos estructurados son datos que tienen relaciones internas bien definidas y restricciones que proveen un contexto para los datos, la mayoría de las veces imprescindible para su correcto procesamiento*. Típicamente, aplicaciones que manipulan datos contienen reglas de negocios que definen el contexto de los datos. Sin embargo, cuando los datos son intercambiados entre aplicaciones, existe la posibilidad de que no sean correctamente interpretados. Por tal razón, las aplicaciones entienden de mejor forma los datos que están estructurados. Ejemplo de esto son los archivos planos en código ASCII, el cual siempre es provisto junto con un documento aparte, el cual se denomina "layout".

Dentro de una organización, tales problemas pueden resolverse en cierta medida al utilizar los *diccionarios de datos y otros estándares*. Desde que surgió Internet, sin embargo, *ha crecido la necesidad de compartir los datos electrónicamente entre organizaciones*. Esto es altamente importante para el **comercio electrónico negocio a negocio (B2B)**.

Para facilitar un intercambio abierto de los datos, se debe definir con algún instrumento la estructura y contenido de los datos a detalle antes de compartirlos con otras aplicaciones. **El instrumento que usa XML es el término "metadata" que denota datos que describen a otros datos**. Los metadata proveen un contexto para los datos que están compartidos entre aplicaciones. Esto implica que se permite ahora que estas aplicaciones interpreten los datos que reciben de una forma significativa.

Los datos son solamente útiles si su contexto es perfectamente comprensible. Un claro ejemplo de esto es representar el precio de un producto con "4". Aquí, esta pieza de información no tiene sentido a menos que se conozca que el precio está expresado en un tipo de moneda en particular (pesos, dólares, yenes, francos, euros, etc.)[154]

Incluir los metadata en los datos estructurados no es el único interés de las organizaciones. La mayoría de las corporaciones y empresas típicamente **deben enfrentar problemas que**, si no se atacan rápidamente, *ocasionan la pérdida de productividad e inflexibilidad en el negocio*. Dichos problemas son[155]:

- **Crear y distribuir información documental de alto valor**. Entre las organizaciones que distribuyen información de alto valor se encuentran publicadores de datos técnicos, médicos, legales, de negocios, financieros o personales. Dichas organizaciones hacen fuertes inversiones en la creación y distribución de información que sus clientes consideran crucial. Como resultado, la organización gana máximos beneficios de los métodos que incrementan la *precisión, seguridad y flexibilidad*, y reducen el tiempo de procesamiento de su información, a la par que se reducen los costos de producción y mantenimiento.
- **Generación de formatos de salida múltiples**. Existe una creciente necesidad para encontrar una vía de generar datos en múltiples formatos tales como documentación impresa, distribución en CDROM, entrega en línea o por Internet, impresión bajo demanda, archivos de ayuda y datos legibles preferentemente por una máquina.
- **Integración de información de múltiples fuentes**. Cuando la información debe ser recogida desde múltiples fuentes, el esfuerzo invertido en integrar dicha información es sustancialmente reducido si todas las fuentes usan un método común de descripción de datos dentro de una comunidad de interés común, en lugar de usar formatos múltiples y variados.
- **Administración de información y combinación condicional de información**. Si los datos dependen de información externa, dicha información podría almacenarse en una variedad de sistemas, en localidades distribuidas, utilizando muchos formatos diferentes. Deben existir medios de localizar, describir, obtener y visualizar mensajes desde dichos sistemas para incorporarlos en los datos.

En vista de las necesidades de información de las empresas, ¿a qué criterios deben apegarse los datos a fin de *solucionar los problemas con su procesamiento*? Los métodos utilizados para organizar y representar la información de negocios deben alcanzar los siguientes requerimientos:

[154] Microsoft Corporation. "Overview of XML Documents" En: Building XML-Based Web Applications, p. 3
[155] Ibid.

- Los datos deben ser legibles tanto por computadoras como por usuarios humanos.
- Tanto el contenido como la estructura de los datos deben estar definidos.
- Las relaciones entre los componentes de datos deben estar enfatizados.
- La estructura de datos debe estar separada de su presentación de los datos.
- La estructura debe estar abierta y ser ampliable

En respuesta a la comunicación de estas necesidades por diversas compañías, un método estandarizado para representar los datos fue propuesto por la organización W3C, y después recomendado para desarrollo por todas las partes interesadas. Este estándar llena todos los requerimientos listados, y ha llegado a ser el método más importante de describir los datos utilizados en la industria al día de hoy. Este estándar es llamado XML[156].

¿Qué es XML? Si las aplicaciones deben procesar sus datos inteligentemente entonces deben entender el contexto y la estructura de los datos. **XML es una sintaxis de marcas que define la estructura de los datos de una forma abierta y auto descriptiva**. XML define meta datos descriptivos aplicando una marca en cada componente de datos a través de una sintaxis muy particular.

Esto permite que los *datos puedan ser transferidos fácilmente sobre una red y procesados consistentemente por las aplicaciones receptoras*. Debido a que XML es utilizado para describir y estructurar información, se puede pensar en XML como un lenguaje de descripción de datos. Se puede usar XML para describir componentes de datos, registros, y otras estructuras de datos complejas tales como órdenes de compra, catálogos y documentos en general.
En la siguiente figura se muestra la misma orden de compra, una con datos no estructurados (por ejemplo, en formato HTML y mostrada en un Navegador) y otra con datos estructurados en XML[157].

Orden de Compra sin marcas

Sociedad de Comercio del Sur SA de CV
Orden de Compra: 567123

30/08/2004

Dirigido a: Grupo Contable y Asociados

Alitas de pollo	20	10
Croquetas	10	15

350

Orden de Compra marcada con etiquetas

```
<OrdenDeCompra numero="567123">
<Origen>Sociedad de Comercio del Sur SA de
CV</Origen>
<Destino>Grupo Contable y Asociados</Destino>
<Fecha>30/08/2004</Fecha>
<Producto>
<Nombre>Alitas de Pollo</Nombre>
<Cantidad>20</Cantidad>
<PrecioUnitario>10</PrecioUnitario>
</Producto>
<Producto>
<Nombre>Croquetas</Nombre>
<Cantidad>10</Cantidad>
<PrecioUnitario>15</PrecioUnitario>
</Producto>
<Total moneda="PesosMexicano">350</Total>
</OrdenDeCompra>
```

Ilustración 31 El lenguaje Extensible Markup Language (XML) define una sintaxis para describir y estructurar los datos

Un documento XML nos recuerda a los documentos HTML. **¿Qué semejanzas y diferencias existen entre ambos lenguajes?** XML es a veces llamado un lenguaje de marcas porque se puede utilizar para definir estructuras de datos al aplicar etiquetas para marcar datos. Sin embargo, se puede definir un conjunto de etiquetas propias que describa los datos en la forma que la organización la encuentre útil. Esto es diferente a HTML, en el cual un conjunto preestablecido de etiquetas describe cómo los datos deben ser presentados en lugar de determinar que es lo que representa cada dato. Por tanto, XML es apropiadamente más considerado un lenguaje meta markup, porque no define un conjunto de etiquetas como lo hacen otros lenguajes, pero en lugar de ello provee las reglas para definir un lenguaje de marcas propio.

[156] Ibid., p. 5
[157] Ibid., p. 7

En esta forma, XML genera un control completo sobre la estructura de los documentos, y la presencia de etiquetas hace fácil para otras aplicaciones acertar en el significado de los datos y procesarlos adecuadamente.

¿Qué hace que XML pueda procesarse tan fácilmente en cualquier aplicación? *La razón principal es que XML está basado en texto*. En este sentido, los datos de XML están retenidos en un formato simple y abierto, por lo que pueden ser interpretado por las aplicaciones, en algunos casos, más fácilmente que un simple archivo plano ASCII. Por otra parte, el hecho de que los documentos XML contienen texto en lugar de datos binarios es otra ventaja clave. Las aplicaciones pueden interpretar un documento XML, buscando etiquetas específicas de interés a dichas aplicaciones, y hasta pueden ignorar libremente etiquetas desconocidas, así como sus datos asociados[158].

XML es sin lugar a dudas, una idea brillante, pero, ¿a quién se le ocurrió primero? XML está basado en el Standard Generalized Markup Language (SGML), el cual fue definido en 1986 por la International Organization for Standardization (ISO) como el estándar ISO 8879. SGML ha sido utilizado desde 1986 a través de un amplio rango de los sectores de la industria y en muchos formatos diferentes, uno de los cuales es HTML.

Los objetivos de aquellos que diseñaron SGML fue simple: confrontar un creciente número de llamados "lenguajes de marcas" para textos electrónicos, con cada lenguaje más o menos apegado a un tipo particular de procesamiento o aún con un paquete de software en particular. Ellos pensaron en definir un simple lenguaje en el cual todos los esquemas pudieran ser expresados, por lo que la información esencial representada por dichos textos pudiera ser transferida desde un programa o aplicación hacia otro.

SGML es un potente estándar para lenguajes de etiquetas. Pero dicho es complejo, y su complejidad ha impedido un uso popular. XML, un subconjunto de SGML, significativamente simplifica SGML, reteniendo los buenos puntos de SGML pero haciéndose en sí mismo fácil de aprender y utilizar por toda la vasta comunidad de desarrolladores[159].

¿Qué relación existe entre los lenguajes de etiquetas SGML, HTML y XML? Existe cierta confusión sobre la relación entre SGML, HTML y XML, por lo que es importante dejar claro que:

- SGML es un lenguaje meta markup,
- HTML es una aplicación de SGML
- XML es un subconjunto de SGML[160]

¿Es la ISO también el organismo regulador de los estándares de XML? No. Actualmente, quien está a cargo de actualizar la norma XML es W3C. XML es un estándar que fue alcanzado a través de la colaboración de muchas compañías diferentes, y a dicho consorcio se le nombró W3C. El site oficial de este es http://www.w3.org/

¿Por qué ha ganado tanta popularidad el trabajo realizado por la W3C? Una de las razones que explican el rápido crecimiento y adopción de XML como una nueva y excitante tecnología es que reconoce que *los documentos consisten de tres componentes distintos*[161]:

- **Datos**. Por ejemplo, un documento de procesador de textos contiene texto, puntuación y espacios en blanco.
- **Estructura**. Por ejemplo, los documentos del procesador de textos tiene una estructura diferente a la de las hojas de cálculo, lo cual en consecuencia tiene una estructura diferente a la de los archivos para presentaciones. La estructura de un documento define el tipo de documento, la organización de sus elementos, y los tipos permitidos y el ordenamiento de los elementos dentro del documento.
- **Presentación**. ¿Cómo se presenta la información al usuario? ¿Qué tipos de letra son utilizados? ¿Qué colores se requieren tanto para el primer plano como para el fondo?

[158] Ibid.
[159] Ibid., p. 8
[160] Ibid.
[161] Ibid., p. 9, 10

La aproximación XML describe solo los datos y su estructura. XML no tiene que ver con la presentación de los datos, a menos que se esté utilizando su rama muy particular de las hojas de estilo. También considera que la estructura inherente de un documento y la información acerca de los tipos de datos en el documento poseen la misma importancia. Tanto la presentación como el formato de la información se mantienen separados del contenido y la información estructural.

¿Cómo se ha amoldado XML al World Wide Web y viceversa, si el Web tradicionalmente ha llegado a ser una colección de páginas formateadas en HTML? La razón es que las páginas HTML y archivos relacionados son transmitidos con el protocolo Hypertext Transfer Protocol (HTTP). Los clientes de Web tales como los navegadores envían solicitudes a través de HTTP a los servidores Web, los cuales interpretan, aceptan y responden dichas solicitudes. HTTP fue específicamente diseñado para transportar HTML. **Debido a que XML es similar a HTML**, este también es ideal para transportarse a través de HTTP[162].

En este sentido, XML no solamente ha permitido un mejor método de comunicación entre aplicaciones. *También permite que diferentes tipos de dispositivos se conecten a la Web*. En el pasado, los navegadores estaban típicamente confinados a las computadoras de escritorio y laptops. Después, diferentes tipos de dispositivos necesitaron obtener información de la World Wide Web, incluyendo computadoras de mano y de bolsillo, así como teléfonos celulares. Debido a las limitaciones de presentación de datos de los dispositivos pequeños, no es posible enviarles datos en el mismo formato en cada tipo de dispositivo.

Generar XML en lugar de HTML permite que los Web servers determinen que tipo de dispositivo realiza la solicitud y responder utilizando el formato de los datos adecuado. Los lenguajes basados en XML tales como el Wireless Markup Language (WML) ha sido creado específicamente para formatear los datos para dispositivos con capacidades de presentación de datos limitadas. El XML es traducido en su lenguaje de presentación apropiado tal como HTML o WML al aplicar una hoja de estilo, ya sea del lado del servidor o bien del lado del cliente.

Los navegadores también han avanzado en la tecnología XML. Al día de hoy pueden encontrarse navegadores "inteligentes" los cuales tienen intérpretes de XML incorporados en su construcción, tales como Microsoft Internet Explorer 6.0. Se dice que son "inteligentes" porque pueden también recibir XML puro para hacer la transformación a HTML u otro XML del lado del cliente. Esto mejora el rendimiento de todos los servidores Web, porque dichos no tienen que realizar la transformación de los datos.

XML no solo reduce la carga en el servidor, sino que también *permite que los navegadores manipulen sus datos XML de una forma "inteligente", debido a que los datos XML contienen etiquetas significativas que pueden ser utilizadas y manipuladas con un script del lado del cliente*.

Esto reduce el número de veces que el navegador debe comunicarse con el servidor para procesar datos o recibir una vista diferente de los datos. Mucho de este trabajo puede hacerse usando un pequeño "script" del lado del cliente[163].

Específicamente, ¿Cuáles son las diferencias entre HTML y XML? Aunque tanto HTML como XML utilizan etiquetas para proveer marcas para los datos en el Web, tienen un número de diferencias importantes.

Como lenguaje de marcado, *HTML provee un conjunto fijo de etiquetas* que pueden utilizarse para dar formato a los datos presentados en el explorador de Internet. Alternativamente, XML es un lenguaje de meta marcado, sin etiquetas de su pertenencia, por lo que se puede utilizar para crear nuevos lenguajes de marcado que **describa los datos en una forma significativa**.

- Los documentos HTML contienen etiquetas que proveen al navegador la información acerca de cómo los datos deben ser mostrados. El conjunto de etiquetas permitido en un documento HTML está bien definido y está fijo. No se pueden definir nuevos tipos de elementos en HTML.

[162] Ibid., p. 11
[163] Ibid.

- XML describe cómo están estructurados los datos, no cómo deben ser mostrados o utilizados. Los documentos XML contienen etiquetas que asignan un significado al contenido del documento. Estas etiquetas permiten que las aplicaciones interpreten los datos y busquen lo que necesitan dentro del documento XML[164].

¿Cuáles son las ventajas de XML? **XML hace fácil intercambiar información dentro de la misma organización o entre diferentes organizaciones**. *XML es también útil como medio de integración de diversas fuentes de información para presentar un producto final de datos en una interfaz uniforme*.

Sin embargo, XML no es la solución general para todos los problemas. Tal como todas las tecnologías, tiene ventajas y desventajas. Las ventajas son[165]:

- **Extensibilidad**. El desarrollador puede utilizar lenguajes de etiquetas existentes o crear los suyos propios.
- **Apertura**. XML no pertenece a alguna compañía en particular y es un estándar manejado por organizaciones no lucrativas.
- **Interoperable**. XML no depende del sistema operativo, lenguaje, o fuente de datos de las aplicaciones que lo usan.
- **Datos autodescritos**. El objetivo con XML es quelos datos sean auto descriptivos por lo que su estructura es fácilmente identificable por las aplicaciones.

En contraparte, ¿Qué situaciones podrían situar en desventaja al lenguaje XML? Se tienen dos escenarios[166]:

- **Múltiples lenguajes de etiquetas pueden ser creados para el mismo propósito**. Debido a que XML es ampliable, se puede crear fácilmente un lenguaje de etiquetado propio para resolver una necesidad de negocios en particular. Sin embargo, esto dirige a la situación de que cada organización tiene su propio conjunto de lenguajes de etiqueta, lo cual, en un momento determinado impide la intercomunicación.
  Dicho problema es manejable en parte al establecerse un abanico de organizaciones no lucrativas que actúan como repositorio de definiciones de lenguajes de etiquetas. Por ejemplo, the Organization for the Advancement of Structured Information Systems (OASIS) mantiene un repositorio de XML en http://www.oasis-open.org/. Similarmente, existe una biblioteca de XML schemas en http://www.biztalk.org/. Se puede también transformar una estructura de documento XML en otra al usar el Extensible Stylesheet Language for Transformation (XSLT).
- **Algunos aspectos de XML todavía no son un estándar**. XML es relativamente, una nueva tecnología, y los estándares de XML aún están evolucionando en ciertas áreas clave. La W3C tiene muchos grupos diferentes que están trabajando en la creación de tecnologías y mejorar los estándares existentes. Por tal razón, siempre existirá una brecha de tiempo entre la aparición de nuevas tecnologías y su adopción por parte de la industria.
  Esta situación mejorará en los próximos años en la medida que los estándares de XML se establezcan y ganen un amplio soporte por diversos vendedores. Mientras tanto, las organizaciones deben hacer decisiones estratégicas acerca de qué aspectos de XML usarán en sus aplicaciones.

Gracias a las ventajas de XML, y a pesar de un par de desventajas a resolver, XML tiene una gama de usos cada vez extensa. La naturaleza abierta y ampliable de XML lo hace extremadamente ideal para muchas y muy diferentes aplicaciones. XML tiene aplicaciones en negocios, ciencia, cuidado de la salud, tecnología de información y multimedia, algunos de los cuales son descritos a continuación:

- **Enterprise Application Integration**. Debido a su fuerte naturaleza estructurada, XML trabaja bien como un mecanismo de intercambio de datos en sistemas distribuidos. Esto significa que se puede utilizar XML para codificar los datos que son transferidos entre muy diferentes sistemas ejecutándose en diferentes sistemas operativos, o software de aplicación. Esto provee la ventaja de permitir que las organizaciones conecten diversas aplicaciones en su línea de negocios. Tales comunicaciones son llamadas Enterprise Application Integration (EAI).
- **Business to Business e-commerce**. Tomando a la Enterprise Application Integration como punto de partida, se tiene que los socios de comercio también pueden intercambiar entre sí documentos de negocios, tales como órdenes de compra, facturas, y notificaciones de embargo a través de Internet, aún si sus sistemas internos son

[164] Ibid., pp. 12, 13
[165] Ibid., p. 14
[166] Ibid., p. 15

incompatibles. Este tipo de intercambio de datos puede reemplazar el área de business-to-business que la Electronic Data Interchange (EDI) ha utilizado durante mucho tiempo. Al utilizar Internet como medio de transporte, XML puede proveer la funcionalidad de EDI en una fracción de su costo.

- **Distribución de información**. Esta aplicación de XML juega un papel importante en la competitividad de muchas organizaciones. En un escenario de negocios típico. La información debe estar distribuida internamente para empleados, externamente para socios de negocios y clientes, y de forma más amplia para el público en general.
  El Web provee una infraestructura conveniente para diseminar diferentes tipos de información. Sin embargo, XML no esta diseñado para distribución de información debido a que no retiene el significado real de los datos, por lo que el procesamiento local de los datos permanece dificultoso. XML abate las limitaciones de HTML. Ligado con el uso de las hojas de estilo XSLT, describen como los datos deben ser presentados. Los documentos XML permiten una distribución de información confiable dentro y fuera de la organización[167].

Esto es lo que se tiene que decir en cuanto a las ventajas, desventajas y usos de XML. **A continuación, se hará una descripción de los aspectos técnicos de los documentos XML. Se detallarán cuestiones básicas de los documentos XML, es decir, los elementos, atributos, gramáticas y namespaces**.

En primer lugar, se tiene que la unidad básica de información en XML es el elemento. Los documentos XML primeramente contienen etiquetas y texto. Las etiquetas definen los elementos de datos, y el texto provee los datos representados en el documento.

La sintaxis básica de un elemento XML es como sigue[168]:

```
<name_of_element>text content</name_of_element>
```

El elemento inicia con la etiqueta de apertura <name_of_element>, y finaliza con la etiqueta de cierre </name_of_element>. Se puede colocar el contenido textual (valor) para el elemento entre la etiqueta de inicio y la etiqueta que cierra.

Por ejemplo, el siguiente elemento XML especifica el nombre de una persona:

```
<nombre>Saul Moreno</nombre>
```

De forma similar, el siguiente elemento XML representa el salario de la persona:

```
<salario>5000</nombre>
```

Los elementos XML pueden, además de contener un valor, uno o más elementos en su contenido. En ese caso se dice que los elementos están anidados. La aplicación del anidamiento en los elementos de XML tiene como objetivo formar una jerarquía de información relacionada. El principio de los elementos anidados y datos textuales es fundamentalmente el mismo en cualquier documento XML, sin importar que tan complejo o grande pueda ser el documento en sí. Esta simplicidad de sintaxis es precisamente lo que da a XML su poder, flexibilidad y apertura[169].

Por ejemplo, el siguiente fragmento XML muestra como agrupar el nombre y el salario de un empleado en un elemento <empleado>:

```
<empleado>
      <nombre>Saul Moreno</nombre>
      <salario>5000</nombre>
</empleado>
```

Un dato notable sobre el anidamiento es que, dentro de un elemento, existen series de elementos repetitivos. Por ejemplo, una compañía podría tener un documento XML que retenga una lista de todos sus empleados. El documento XML podría aparecer como sigue:

```
<empleados>
```

[167] Ibid., pp. 16, 17
[168] Ibid., p. 19
[169] Ibid.

```
      <empleado>
            <nombre>Alberto Aparicio</nombre>
            <salario>18000</salario>
      </empleado>
      <empleado>
            <nombre>Ernesto Santos</nombre>
            <salario>15000</salario>
      </empleado>
      <empleado>
            <nombre>Saul Moreno</nombre>
            <salario>7000</salario>
      </empleado>
</empleados>
```

Este ejemplo utiliza un elemento llamado `<empleados>` para agrupar todos los empleados. Existen tres empleados, cada uno representado por un elemento `<empleado>`. Cada empleado tiene los elementos `<nombre>` y `<salario>`.

¿Qué otras estructuras de datos están contenidas en un documento XML? *Los elementos son el corazón de un documento XML, pero no son el único tipo de información que pueden aparecer en un documento XML. También existen*[170]:

- **Declaraciones de XML**. La mayoría de los documentos XML inician con una construcción especial llamada la declaración. Esta declaración opcional especifica que el resto del documento debe ser tratado por una aplicación que procese dicho _XML. La sintaxis es como sigue:
  `<?xml version="1.0" standalone="yes/no" encoding="encoding"?>`
  El único atributo requerido es el atributo version, el cual debe tener el valor de "1.0". Los otros atributos son opcionales. El atributo stand-alone especifica si otros archivos son requeridos para mostrar el documento completo, tal como un Document Type Definition (DTD). El atributo encoding es muy importante desde la perspectiva de la globalización, porque dicho especifica el conjunto de caracteres que se utilizan en el documento. Los valores comunes de este atributo son listados en la siguiente tabla:

Encoding	Description
ISO-8859-1	Latin alphabet 1. Used for most western European Languages. Similar to 8-bit ASCII
UTF-8	8-bit Unicode equivalent to ISO-8859-1
UTF-16	16-bit Unicode used for international character sets

- **Instrucciones de procesamiento**. Las instrucciones de procesamiento pueden también aparecer en un documento XML.Si están presentes en un documento, la instrucción de procesamiento inicia con un signo menor que y un signo de interrogación que cierra (**<?**) y finaliza con un signo de interrogación que cierra y un mayor que (**?>**), similar a la declaración XML. Sin embargo, la declaración XML no es un ejemplo de una instrucción de procesamiento.
  Las instrucciones de procesamiento proveen información extra al intérprete de XML para ayudar a procesar correctamente el contenido del documento. Por ejemplo, se puede utilizar la siguiente instrucción de procesamiento para aplicar una hoja de estilo al documento, para transformar su apariencia en el Web Browser.
  `<?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>`
- **Comentarios**. Opcionalmente se pueden colocar comentarios en cualquier parte en un documento XML excepto dentro de las etiquetas de inicio y fin. Los comentarios inician con **<!--** y finalizan con **-->**, e incluso pueden abarcar diversas líneas. En un comentario se tiene la restricción de no poder utilizar la cadena de guión doble**--.**
- **Elementos**. Los elementos son los componentes básicos de un documento XML. Los elementos ocurren en dos variedades: aquellos que tienen algún tipo de contenido y aquellos que se encuentran vacíos.
  El siguiente es un ejemplo de un elemento que contiene texto. El elemento <nombre> retiene el nombre de una persona:
  `<nombre>Saul Moreno</nombre>`
  El elemento inicia con una etiqueta `<nombre>` que abre. A continuación de esta etiqueta, se está libre de definir algún dato de caracteres que represente el nombre de la persona. El elemento debe estar terminado con una etiqueta

[170] Ibid., p. 21

de cierre </nombre>. Si se llega a omitir esta última etiqueta, un error ocurre cuando el documento XML es analizado.

- **Elementos hijos**. Los datos en un documento XML son jerárquicos. Los elementos pueden contener otros elementos que son llamados elementos hijos.
  En el siguiente XML, el elemento <nombre> es un elemento hijo de <empleado>:
  ```
  <empleado>
        <nombre>Saul Moreno</nombre>
  </empleado>
  ```
- **Elementos vacíos**. Un elemento vacío es aquel que no contiene texto u otro elemento hijo. Si se tiene un elemento que no necesita algún texto asociado o elementos hijos –lo que comúnmente ocurre en las bases de datos a la hora de representar un campo con valor Nulo-, una forma de definir el elemento es como sigue:
  ```
  <empleado_planta></empleado_planta>
  ```
  Se puede utilizar también la siguiente sintaxis simplificada para elementos vacíos. Se debe destacar el slash al final de la etiqueta. Esto indica que la etiqueta representa el inicio y el fin del elemento, por lo que el elemento es vacío:
  ```
  <empleado_planta/>
  ```

- **Atributos**. Se pueden utilizar atributos para proveer información adicional sobre un elemento. Los atributos deben ser definidos en la etiqueta de inicio de un elemento, como se muestra en el siguiente ejemplo:
  ```
  <salario moneda="PesosMexicanos">7000</salario>
  ```
  Este ejemplo define un elemento llamado <salario> y especifica que el salario de la persona es 7000. El elemento salario tiene el atributo llamado moneda que provee información adicional sobre el salario de la persona. El atributo especifica que el salario está expresado en pesos mexicanos.
  Los valores en los atributos deben estar encerrados entre comillas. Se pueden usar apóstrofes 'PesosMexicanos' o comillas dobles "PesosMexicanos" siempre y cuando existan en pares.

Hasta aquí aparecen todas las piezas del rompecabezas sobre la mesa pero, ¿cómo viene ensamblado el documento XML? ¿Cuál es su estructura, formalmente hablando? La especificación de XML, localizada en Internet en http://www.w3.org/TR/REC-xml define la estructura de un documento XML bien formado. La estructura básica contiene un prólogo y un elemento raíz, como sigue[171]:

- **Prólogo**. Contiene la declaración de XML e instrucciones de procesamiento. También contiene la gramática a través de un documento DTD o de un XML Schema.
  El prólogo contiene metadata acerca del resto del documento, tal como la versión de la especificación de XML con la que cumple, y la codificación de caracteres utilizada en el resto del documento. Esta también contiene instrucciones de procesamiento utilizadas por las aplicaciones intérprete.
  El prólogo puede incluir DTDs o XML Schemas, cada uno de los cuales describe la estructura del resto del documento en términos de cómo los elementos y atributos se relacionan entre sí (por ejemplo, cuáles atributos pueden ser utilizados en qué elementos)
  A continuación se muestra un ejemplo del prólogo:
  ```
  <?xml version="1.0"?>
  <!DOCTYPE employees SYSTEM "empleados.dtd">
  ```
- **El elemento root**. Contiene todos los demás datos. En una especificación formal de un documento XML, los datos dependen de un elementos simples llamado el elemento root. El elemento root contiene todos los demás elementos y atributos en el documento. Por ejemplo:
  ```
  <employee>
     <nombre>Saul Moreno</nombre>
  </employee>
  ```

Hasta aquí es fácil entender el prólogo del documento XML, pero ¿cómo puede revisarse si el elemento root está bien formado? Esta pregunta es importante porque si el documento XML no está bien formado, no podría ser analizado por el intérprete. Para que un documento XML sea bien formado, deben ser obedecidas las siguientes reglas[172]:

[171] Ibid., p. 23
[172] Ibid., p. 24

- **Elemento root simple**. Solamente puede existir un elemento root en un documento XML.Todos los demás elementos deben estar definidos dentro del elemento rot.
- **Etiquetas de apertura y cierre en correspondencia**. XML requiere que cada etiqueta que abre tenga su etiqueta que cierra correspondiente. Si se desea crear un elemento vacío, se puede utilizar la sintaxis <element_name/> como una abreviatura de <element_name></element_name>.
- **Mayúsculas y minúsculas consistentes**. XML es sensitivo a mayúsculas y minúsculas. Por ejemplo, si se tiene una etiqueta inicial llamada <empleado>, se debe tener una etiqueta </empleado> debido a que si se utiliza una etiqueta diferente, por ejemplo </Empleado>, los elementos no tendrán correspondencia alguna.
- **Elementos correctamente anidados (sin traslape de elementos)**. Los elementos deben estar completamente encerrados por sus elementos padre.Porejemplo elsiguiente ejemplo se encuentra bien formado porque el elemento <B> está completamente anidado dentro del elemento <A>:
  ```
  <A>
      <B>
      </B>
  </A>
  ```
  Sin embargo, el siguiente ejemplo no está bien formado porque <B> se traslapa al final del elemento <A>:
  ```
  <A>
      <B>
  </A>
      </B>
  ```
- **Valores de los atributos encerrados en comillas ó apóstrofes**. El valor de los atributos deben estar encerrados en comillas o apóstrofes, siempre y cuando se utilicen de forma consistente.
- **No deben existir atributos repetidos dentro de un elemento**. Si desea representar datos repetitivos, tales como múltiples autores por libro, se debe representar esta información en elementos, no atributos, debido a que todos los atributos dentro de un elemento deben tener nombres únicos.
- **Debe estar compuesto de un conjunto de caracteres valido**. Si desea utilizarse la eñe, el acento, o cualquier otro símbolo de caracteres que no pertenezca al conjunto de caracteres ISO, debe especificarse en el prologo que se está utilizando un conjunto de caracteres 8 bit o 16 bit Unicode.

En un solo día, una empresa puede llegar a recibir muchas decenas, o algunos cientos, incluso hasta unos pocos miles de documentos XML de manera electrónica, para su pronto procesamiento en los sistemas. **¿Cómo determinar si todos estos documentos XML son válidos, sobre todo si son del mismo tipo? Un documento XML válido es aquel que va de acuerdo con un DTD o XML Schema, lo que restringe el contenido permitido en un documento**. El DTD o XML Schema define los elementos y atributos permitidos, el tipo de datos que aquellos elementos y atributos pueden contener, y las relaciones entre los elementos y los atributos[173].

En vista de que un conjunto de información puede adaptarse a diversas estructuras XML, *¿qué factores se deben tomar en cuenta al momento de diseñar un documento XML?* Cuando se define la estructura del documento XML, se debe decidir cómo serán representados los datos. Podrían ser transportados en los valores de atributo o en el contenido de los elementos. Esto es similar a desarrollar una base de datos.

Existe una amplia discusión de este punto, resultando en algunas guías simples que pueden ayudar. Sin embargo, no existen reglas fijas en el diseño de software, por lo que se debe diseñar los XML de manera que se ajusten a la aplicación en particular[174].

Por ejemplo, se pueden tener en cuenta algunos *lineamientos al decidir si un dato será representado con elementos o atributos*[175].

[173] Ibid.
[174] Ibid., p. 32
[175] Ibid.

Se deben utilizar **elementos** para:

- Datos que deberán ser leídos tanto por máquinas como por seres humanos.
- Datos repetitivos, por ejemplo, se desearía usar elementos para almacenar múltiples autores.
- Conjuntos de datos jerárquicos.
- Datos ordenados.

Se podrían utilizar **atributos** para:

- Los datos que deberán ser leídos exclusivamente por máquinas.
- Meta datos para datos acerca de los elementos.
- Valores enumerados.
- Optimización en el tamaño de los documentos XML.

Al poder tomar varias estructuras y valores el lenguaje XML, es necesario delimitar que subconjunto del lenguaje utilizará nuestra aplicación particular. Esto se lleva a cabo mediante una gramática. En varias industrias y sectores, las organizaciones han utilizado XML para crear conjuntos de elementos y atributos para definir los datos que utilizan. Estos conjuntos son comúnmente llamados gramáticas XML[176].

Una gramática XML es un predefinido conjunto de elementos y atributos que son típicamente representados utilizando un DTD o un XML Schema. La siguiente ilustración muestra algunas de las gramáticas XML que están disponibles para dirigir las necesidades específicas de negocios, industria y sectores de entretenimiento.

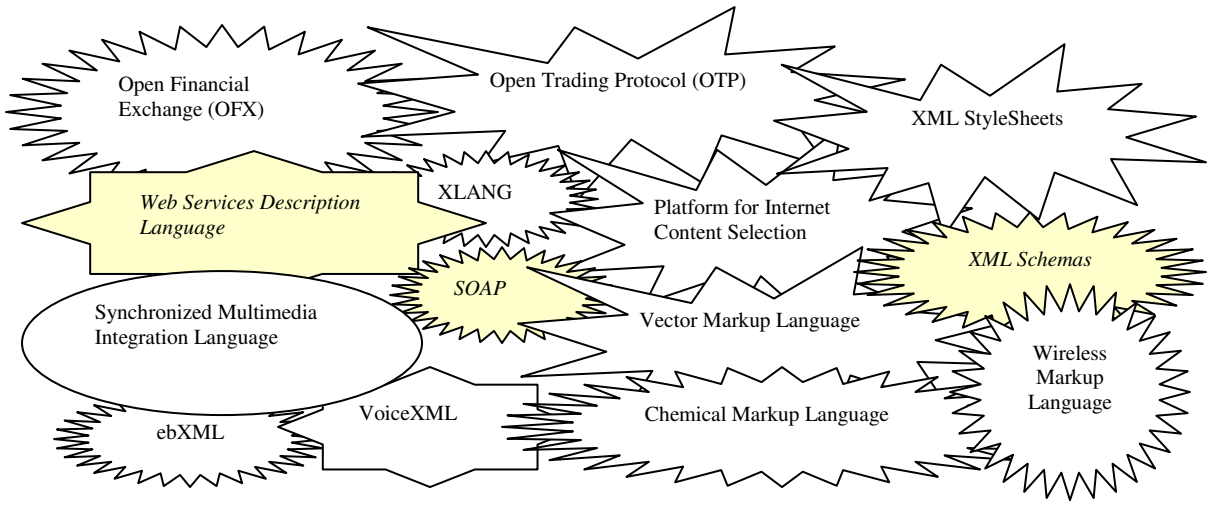


Ilustración 32 Muchas gramáticas XML han sido definidas para diferentes tareas o mercados

Para conocer la lista completa de las gramáticas, se pueden visitar los Web sites:

- http://www.w3.org/xml/
- http://www.oasis-open.org/cover/siteIndex.html
- http://www.biztalk.org/

¿Para qué sirven todas esas gramáticas? ¿Qué aplicaciones tienen?

- **Open Financial Exchange (OFX)** es una especificación de tipos de elementos estándar para representar información financiera tal como números de tarjetas de crédito, fechas de expiración, tipos de transacciones, y así sucesivamente. El objetivo es que la aplicación Web del cliente reciba los datos en el navegador utilizando el

[176] Ibid. p. 34

medio del HTML convencional, y entonces presentar los datos a instituciones financieras en el estándar de la industria OFX XML. Sobre este estándar puede consultarse información en http://www.ofx.net/

- **Open Trading Protocol (OTP)** es una amplia iniciativa diseñada para permitir que diversas tecnologías de e-commerce coexistan e ínter operen de una forma suave. OTP usa XML para describir la estructura y contenido de varios tipos de mensajes necesitados para ejecutar una transacción electrónica. Al hacer esto, OTP provee un estándar abierto que habilita los vendedores de e-commerce, compradores y proveedores el comunicarse en el mismo lenguaje[177]. Mayores detalles acerca de OTP se encuentran en el sitio Web http://ww.ietf.org/html.charters/trade-charter.html
- Las gramáticas utilizadas en el Web service con aplicaciones a la industria automotriz del presente trabajo de investigación son **XML Schema (XSD)**, **Web Services Description Language (WSDL)** y **Simple Object Access Protocol (SOAP).** Dichas serán propiamente tratadas con todo detalle en las secciones **3.2.1**, **3.2.2** y **3.2.3** de la presente tesis.

¿Qué preguntas tuvieron que hacerse los diseñadores de esas gramáticas para poder representar la información que requerían de una forma adecuada? ¿Qué reglas deben definirse antes de crear una "plantilla con dichas reglas"? Las preguntas son las siguientes[178]:

- **Elementos**. ¿Cuál es el nombre de los elementos? ¿Qué relación debe existir entre dichos elementos? ¿Qué tipos de datos deben contener los elementos?
- **Atributos**. ¿Cuáles son los nombres de los atributos? ¿Los atributos son opcionales o requeridos? ¿Qué tipos de datos deben contener los atributos?
- **Entidades**. ¿Qué texto se espera en los elementos? ¿Es posible incluir sintaxis de otros documentos en la nueva definición de XML?

Anteriormente se habló de una "plantilla con reglas" ¿Cómo debe crearse dicha plantilla? Más propiamente hablando, ¿cuantas formas de definir una gramática XML existen? XML es ampliable o extensible, esto es, si no se puede encontrar una gramática existente que alcance nuestras necesidades, se puede desarrollar una nueva gramática que contenga una mezcla de elementos y atributos que nuestra aplicación requiera.

Existen dos formas básicas de definir la gramática para nuestros documentos XML[179]:

- **Crear un DTD que defina los elementos y atributos permitidos en el documento XML**. El DTD también especifica las relaciones entre los elementos y atributos. Los DTDs fueron los medios originales de definir las gramáticas XML como está delineado en la recomendación del W3C XML.
- **Crear un XML schema**. Los XML schemas tienen el mismo propósito de los DTDs, pero se pueden usar para especificar la gramática de una forma más rigurosa y para definir un conjunto de tipos de datos mas rico. Los XML Schemas son la recomendación de W3C y son el método preferido de definir las gramáticas de XML.

Tal como una escuadra restringe el movimiento de un lápiz al trazar un juego de líneas, las gramáticas de XML restringen el contenido de los documentos instancia de XML. Tanto los DTDs como los XML Schemas permiten establecer restricciones en:

- Los nombres de los elementos.
- Las relaciones existentes entre dichos elementos.
- Los nombres de los atributos.
- Si los atributos son opcionales o requeridos.
- Qué tipos de datos deben contener los elementos y los atributos.

[177] Ibid.
[178] Ibid., p. 35
[179] Ibid.

Los documentos XML pueden también contener entidades, las cuales son elementos de información representados utilizando una notación abreviada, y apuntan a recursos externos. También actúan como representantes de otras piezas de información. Las entidades son típicamente declaradas en un DTD y pueden también formar parte de una definición en una gramática[180].

Antes de pasar al siguiente apartado, debe tenerse claro en la mente el concepto de namespace en un documento XML. Si se puede crear una gramática XML propia al utilizar elementos y atributos definidos por nuestra organización, entonces es común que se llegue a definir nombres de elementos y atributos que ya se están utilizando en otras gramáticas. Si debemos combinar datos de múltiples fuentes, entonces es posible que se tenga un número de elementos y atributos que usan los mismos nombres pero signifiquen diferentes cosas. Esto se conoce como colisión.

Un namespace de XML es una colección lógica de nombres de elementos y atributos que deben garantizarse que sean únicos, poniéndoles un "alias", un "apellido" o una "calificación" con un Uniform Resource Identifier (**URI**). Aún si dos elementos tienen el mismo nombre, si estos pertenecen a namespaces con diferentes URIs, un intérprete diseñado para procesar los namespaces puede diferenciar los dos elementos perfectamente[181].

¿Cómo se etiquetan los namespaces? **Un URI puede ser un Uniform Resource Locator (URL)**, tal como http://www.unam.mx/ServicioHTTP, o bien un **Uniform Resource Name (URN)**, por ejemplo urn:schemas-java-com:xml-data. El path al cual apuntan los URL no tiene porque tener algún contenido en específico, o siquiera existir en Internet, sino que solamente es una cadena única, aunque el URL de la organización una buena opción[182].

Antes de usar un namespace, es esencial declararlo. Los namespaces son declarados utilizando el atributo reservado **xmlns**. Los namespaces pueden ser declarados en algún elemento de un documento XML. En tal caso, cualquier elemento hijo de dicho elemento se encontrará automáticamente incluido en el alcance de dicho namespace a menos que otro namespace sea declarado en el nivel del nodo hijo, el nodo nieto, o un nivel inferior.

Por ejemplo, se podría declarar un namespace en el elemento <empleado> para especificar que dicho elemento y todos sus hijos pertenecen a una gramática que representa información de nómina al utilizar la siguiente sintaxis:

```
<empleado xmlns="http://www.unam.mx/nomina">
…
</empleado>
```

¿Qué pasa en el caso de que se tengan elementos y atributos provenientes de diferentes gramáticas mezclados en un mismo documento de instancia? ¿Cómo pueden entonces, **declararse múltiples namespaces en un documento XML**? Si se desea especificar que algunos elementos hijos pertenecen a un namespace diferente, se puede declarar un nuevo namespace en cada uno de los elementos hijos. Por ejemplo, se podría declarar un namespace en el elemento <alumno> pero entonces podría existir un elemento hijo llamado <cuenta> que pertenezca a una gramática diferente, identificada por otro namespace, usando las siguientes etiquetas:

```
<alumno xmlns="http://www.unam.mx/nomina">
      <nombre>Saul Moreno</nombre>
      <cuenta xmlns="http://www.unam.mx/cuentas">092018369</cuenta>
</alumno>
```

Para este ejemplo,<nombre> pertenece al mismo namespace de <alumno> debido a que es elemento hijo de ese nodo y está dentro del alcance del namespace "http://www.unam.mx/nomina". Sin embargo, el elemento <cuenta> está en un namespace diferente debido a que tiene su propia declaración de namespace que usa un diferente URI.

[180] Ibid., p. 36
[181] Ibid., p. 37
[182] Ibid.

Cuando un namespace es declarado en esta forma, todos los elementos hijos automáticamente caen en el alcance de dicho namespace. Esta característica de XML es conocida como su default namespace[183].

Habrá casos extremos en que el default namespace no será un recurso suficiente sino que será obligatorio aplicar un namespace explícito en un documento XML. ¿Cómo puede hacerse esto? Aunque se pueden declarar default namespaces que pueden ser aplicados a todos los elementos hijos automáticamente, es usualmente necesario mezclar elementos de diferentes namespaces en el mismo nivel. También, se podría necesitar identificar un atributo perteneciente a un namespace en particular, y esto no es posible cuando se utiliza una declaración de namespace por default. Para hacer una declaración de namespace explícita, se puede asociar el URI del namespace con un prefijo que se anticipa los elementos y atributos como sea necesario.

El método suena muy sencillo, pero ¿cómo puede ilustrarse este caso? Una forma de combinar elementos de diferentes gramáticas es como sigue:

Por ejemplo, un documento XML podría contener elementos y atributos que están definidos para usarse por deferentes partes del negocio. Por ejemplo, el departamento de contabilidad podría tener un elemento <pago> que representa el pago recibido por un cliente:

```
<pago xmlns="http://www.unam.mx/contabilidad">
      <cliente>8951</cliente>
      <cantidad>1000.00</cantidad>
      <fecha_recepcion>30-08-2003</ fecha_recepcion >
</pago>
```

El departamento de nómina podría tener un diferente elemento,llamado también <pago>, representando el pago que se le hace a un empleado por gastos de viáticos:

```
<pago xmlns="http://www.unam.mx/nomina">
      <empleado>Saul Moreno</empleado>
      <departamento>Tecnologías de Información</departamento>
</pago>
```

Si se desea combinar información desde ambas gramáticas en el mismo nivel, debe hacerse declaración del namespace explícita al utilizar URIs que son mapeados a prefijos. Cuando un elemento o nombre de atributo, es llamado nombre calificado o **Qname**. Laparte del nombre sin el prefijo es llamado el **local-name**.

El siguiente ejemplo muestra como declarar un namespace explícito para un elemento XML[184]:

```
<prefijo:local-name xmlns:prefijo="namespaceURI">
…
</prefijo:local-name>
```

¿Cómo pueden resumirse las reglas de los namespaces? Cuando se declara un namespace explícito, solamente los elementos que son hijos de los elementos en los cuales el namespace fue declarado y que tienen el prefijo que apunta al URI, serán asociados con dicho namespace. Los elementos y atributos que no tienen un prefijo no serán asociados con algún namespace a menos que exista un default namespace en el alcance de dicho elemento o atributo.

Algún atributo declarado en un elemento que está dentro del alcance del default namespace no será asociado con ese o algún otro namespace al menos que tenga un prefijo que mapea al URI del namespace. Por esta razón, no es posible asociar un atributo con su default namespace[185].

[183] Ibid.
[184] Ibid., pp. 39, 40
[185] Ibid., p. 39

El siguiente ejemplo muestra un documento XML que contiene elementos de dos diferentes namespaces

```
<acc:pago xmlns:acc="http://www.unam.mx/contabilidad">
      <acc:cliente>8951</acc:cliente>
      <acc:cantidad>1000.00</acc:cantidad>
      <acc:fecha_recepcion>30-08-2003</acc:fecha_recepcion >
      <pr:empleado xmlns:pr="http://www.unam.mx/nomina">
            Saul Moreno
      </pr:empleado>
</acc:pago>
```

A pesar que los namespaces a primera vista podrían verse como un tema difícil y engorroso de comprenderse, alegrará al lector saber que al estudiarse a fondo es posible comprender el trasfondo y la filosofía del lenguaje XML. *En un documento se pueden tener elementos provenientes de diversas gramáticas, y pueden incluso ser validados a partir de las reglas de sus gramáticas de orígen*. Los namespaces son conceptos esenciales para comprender XML y sus tecnologías relacionadas

Por ejemplo, cuando se crea una hoja de estilo XSLT que transforma un XML en HTML,se debe calificar las instrucciones XSLT con el namespace de XSLT predefinido como sigue[186]:

```
<xsl:template match="empleado">
      <html>
      <xsl:apply-templates/>
      </html>
</xsl:template>
```

En este caso, los elementos `<xsl:template>` y `<xsl:apply-templates/>` pertenecen al namespace identificado por el URI http://www.w3.org/1999/XSL/Transform. El elemento `<html>` no pertenece a dicho namespace porque no tiene el prefijo **xsl:**

El estudio concienzudo de los XML namespaces está fuera de los objetivos de la presente tesis, pero si se desea conocer con todo detalle este tema, un buen comienzo es consultar de forma libre en Internet:

- el Web site D VInt Productions, http://www.xml.dvint.com
- xml-schemas1.ppt y xml-schemas2.ppt de Robert Roger L. Costello en XML Technologies Course (búsquese en Google porque este material no tiene Web site fijo)

3.2.1 XML Schemas: Uso de XML para descripción de otros XML.

XML permite compartir información en una forma significativa. XML podría ser utilizado para compartir datos entre aplicaciones, sitios Web, departamentos de una misma empresa o negocios de una misma corporación. En todos los casos, XML provee meta datos descriptivos en la forma de elementos y atributos. Sin embargo, para que la información sea interpretada correctamente, se podría necesitar información adicional para describir como esos meta datos están construidos, y cómo las piezas de datos deben ser manejadas.

Si se parte de la premisa de que XML es extensible, es decir, que no existe un estricto conjunto de elementos y atributos que deben ser usados en todos los documentos XML, encontramos que *se pueden inventar elementos y atributos para satisfacer las necesidades de determinado tipo de documento. Las organizaciones podrían acordar entre ellas una gramática XML que defina un conjunto compartido de elementos y atributos que pueden ser utilizados para un tipo de documento en particular*[187].

[186] Ibid.
[187] Microsoft Corporation. "Validating XML Using Schemas" En: Building XML-Based Web Applications, p. 3

¿Qué criterios están definidos en la gramática XML? **Las gramáticas XML especifican criterios** que los documentos deben seguir. Por ejemplo, una gramática puede definir:

- El conjunto de elementos y atributos permitidos en un documento.
- El orden en el cual los elementos deben aparecer.
- Las relaciones permitidas entre diferentes elementos.
- El número máximo y mínimo de veces que un elemento puede aparecer.
- La lista de atributos permitidos para un elemento.

Adicionalmente, una gramática puede imponer restricciones sobre piezas de datos individuales que aparecen como valores de elemento o atributo, para asegurarse que se amoldan a tipos de datos definidos. Por ejemplo, una gramática puede especificar que todas las fechas deben estar en el formato "**YYYY-DD-MM**", o que todos los valores numéricos deben ser presentados con dos lugares decimales[188].

¿En dónde se utiliza la validación de documentos XML? **Los documentos XML experimentan validación para revisar que cumplen las reglas de una gramática particular**. *La validación asegura que una aplicación u organización receptora puede determinar si un documento es una instancia[189] legítima de dicha gramática*. Las reglas de validación pueden estar definidas utilizando un Document Type Definition (DTD) o un XML Schema.

- Por ejemplo, si dos organizaciones que están involucradas en un negocio electrónico deben intercambiar facturas, podrían acordar en un formato común para esas facturas y definir el formato al crear un XML Schema. Si una de las organizaciones recibe una factura de la otra, pueden realizar validación sobre el documento para asegurarse que cumple con las reglas especificadas en el XML Schema. La validación podrá realizarse con un motor intérprete de validación preconstruido, como Microsoft XML Core Services 4, Microsoft .NET Framework, Multi Schema Validator for Java (msv.jar), etc.
- La validación es importante en los escenarios negocio a negocio porque el procesamiento automático de la factura por la aplicación podría generar errores internos si la estructura no es lo que se espera. Similarmente, si los datos tienen que ser separados para su inserción en una base de datos, podrían surgir problemas si algunas piezas de datos no cumplen con el tipo de datos requerido por las definiciones de campo.

¿Cuáles son los **elementos necesarios para validar un documento XML**? Si es necesario validar una instancia de documento XML para asegurar que su estructura se adapta a su gramática definida, o que cada pieza de datos en el documento es del tipo correcto, se necesitará tener **otro documento que defina las reglas de esa gramática**. Un intérprete de validación podrá leer esas reglas y revisar que el documento de instancia las cumple. El conjunto de reglas puede estar escrito en DTDs o en XML Schemas[190].

Ese otro documento es conocido en el mundo de la programación como **XML Schema**. Un XML Schema es un documento XML que define las restricciones de estructura y tipo de datos para una gramática particular. Las estructuras utilizadas para crear los XML Schemas están definidas por la W3C en http://www.w3.org/TR/xmlschema-1/. Un número de tipos de datos básicos están también definidos por la W3C en http://www.w3.org/TR/xmlschema-2/. Los XML schemas pueden usar esos tipos de datos para restringir el contenido de los elementos y atributos, y un nuevo tipo de datos puede derivarse de ellos como sea necesario.

Debido a que los documentos XML pueden consistir de contenido proveniente de múltiples fuentes, típicamente se trataría de prevenir que los elementos y atributos de una gramática choquen con los elementos y atributos de otra gramática. Para hacer esto posible, usualmente se debe especificar que un grupo de elementos y atributos pertenecen a la misma gramática al calificarlos con un namespace.

---

[188] Ibid.

[189] Una instancia es un ejemplo de un caso particular, por ejemplo, Facultad de Ingeniería es una instancia de las dependencias de la UNAM. También 0155, 0133, 0181 son instancias del campo Lada.

[190] Microsoft Corporation. Validating XML Using Schemas..., p. 5

Las definiciones de namespace son creadas sobre un elemento usando el atributo **xmlns**. El valor de dicho atributo es un Uniform Resource Identifier (URI) que debe identificar de forma única al namespace. Debido a que los URIs son largos y pesados, usualmente se mapea el URI a un prefijo corto y más conveniente[191].

Los XML Schemas son indispensables desde el momento en que dos o más compañías se involucran en la conducción de negocios electrónicos y deciden definir estándares de datos y procesos. Estas deben acordar en crear una gramática, por ejemplo, para órdenes de compra y facturas, y en este caso se tiene que proveer los XML Schemas necesarios para representar esos formatos en los documentos.

Si dos compañías están utilizando un formato de documento común, entonces pueden crear aplicaciones que **procesen automáticamente** los documentos al grado que los datos estructurados apunten hacia campos en la base de datos. Estas aplicaciones parten del hecho que cada instancia de documento se adhiere perfectamente a dicho formato.

Cada vez que una de las compañías envía una factura a otra, el buzón de la compañía puede revisar la estructura de la factura al **validarla contra el XML Schema definido en los estándares**. La compañía puede también revisar que los datos contenidos en el documento no violen las restricciones de tipos de datos impuestas por el schema.

En vista de la importancia de los XML Schemas, *¿Qué lineamientos deben seguirse para crear la estructura de los XML Schemas?* Para crear un XML Schema que defina las reglas en un motor de validación, se deben seguir las estructuras y tipos de datos definidos por la W3C en sus recomendaciones. Las estructuras nos permiten declarar los elementos y atributos que aparecerán en nuestro documento y las relaciones que deben tener entre sí. Los tipos de datos describen tipos de datos preconstruidos que todos los procesadores de schemas deben reconocer. Aquellos nos permiten especificar el tipo de contenido que cada elemento y atributo debe tener, y también declarar nuestros propios tipos que están basados en los tipos de datos preconstruidos.

A continuación se darán las reglas básicas para trabajar con los XML Schemas[192]:

- Para asegurarse de que un intérprete de validación trabaja correctamente con las estructuras del schema y tipos de datos preconstruidos que se pueden utilizar, se debe usar un namespace de la W3C apropiado. Para los XML Schemas, el URI correcto de tal namespace es:
  `http://www.w3.org/2001/XMLSchema`
  Tal como ocurre en los namespaces, este URI de namespace es sensible a mayúsculas y minúsculas. Algun elemento XML, o atributo, o valor que está calificado con este URI del namespace será tratado por el intérprete de validación como una estructura del XML Schema o tipo de datos predefinido.
- El elemento root de cada XML Schema debe ser **schema**. Por tanto, para asegurarse de que está siendo interpretado como el elemento raíz de un XML Schema, se debe calificar este elemento con el URI del namespace apropiado. En el siguiente ejemplo, el elemento schema está calificado con el prefijo **xs:** el cual está mapeado en correspondencia al URI del namespace para los XML Schemas definidos por W3C:
  ```
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ..en este espacio se encuentran las definiciones del schema
   </xs:schema>
  ```
- Documentos xml que no utilizan namespaces. Muchos documentos XML no utilizan declaraciones de namespae, por lo que un schema con este elemento **root** podría describir un documento cuyos elementos y atributos no pertenecen a algún namespace. En tales casos, la única declaración de namespace requerida es la que califica las estructuras y tipos de datos del schema.

Por otra parte, el schema debe definir el namespace de la gramática que valida. Si los documentos descritos por un schema emplean un namespace en particular, los elementos, atributos y tipos de datos definidos por el usuario y declarados en el schema deben estar colocados en dicho namespace. Para hacer esto, se debe configurar una referencia al namespace para el cual es definido el elemento, atributo y declaración de tipo de datos.

[191] Ibid.
[192] Ibid., p. 7

Por ejemplo, para especificar que los elementos, atributos y tipos de datos definidos por el usuario declarados en un XML schema deben pertenecer al namespace representado por el URI `http://tempuri.org/example`, se debe usar el atributo **targetNamespace** en el elemento **schema** como sigue:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://tempuri.org/example"
xmlns:ex="http://tempuri.org/example">
..en este espacio se encuentran las definiciones del schema
 </xs:schema>
```

Una aplicación de procesamiento de schemas no reconoce que este schema esta destinado a declarar elementos, atributos y tipos de datos definidos por el usuario que pertenecen al namespace `http://tempuri.org/example`. Debe destacarse que un nuevo namespace también ha sido declarado y es igual que el target namespace. Esto asegura que todos los elementos, atributos y tipos de datos definidos por el usuario declarados por este schema explícitamente vivirán en el namespace para el cual han sido destinados.

Esta nueva definición de namespace ha sido mapeada al prefijo ex: lo que significa que todas las referencias a elementos, atributos o tipos de datos definidos por el usuario deberán tener el prefijo ex: Sin embargo, debido a que el namespace de los XML Schemas definidos por la W3C han sido declarados por default, todas las estructuras del schema y los tipos de datos preconstruidos no tendrán porque tener prefijos. Esto puede hacer en ocasiones al schema fácil de leer, pero no existe una regla especifica acerca de que namespace deberá quedar como default[193].

¿Qué metodologías deben seguirse para definir la ubicación del Schema en la instancia de XML? Si se recibe un documento XML que se desea validar, se puede usar un schema existente que se obtiene de antemano, o se puede examinar el documento para revisar si contiene una referencia explicita a algún schema. La recomendación de la W2C provee una forma para que los documentos XML apunten a la localidad de dicho schema.

Un documento XML puede informar a un procesador que se encuentra en una instancia de un schema en particular al hacer lo siguiente:

- Referenciar al namespace de los XML Schema instances definido por W3C: http://www.w3.org/2001/XMLSchema-instance
- Definir el atributo **schemaLocation** en ese namespace si el documento usa un namespace.
- Definir el atributo **noNamespaceSchemaLocation** en ese namespace si el documento no utiliza un namespace.

Si el documento con un namespace debe indicar la localidad del schema, debe en primer lugar declarar el namespace de XML Schema-instance definido por la W3C. Este namespace contiene un atributo llamado schemaLocation que puede ser utilizado para asociar el namespace del documento instancia con un schema para ese namespace[194].

¿Qué significa validar un XML mediante otro XML de tipo Schema manualmente? Supóngase que se tiene un documento XML que describe el registro de un empleado y que esta asociado con el namespace `http://hr/empleados` Si el schema que define dicho namespace es localizado en el mismo directorio del documento de instancia, y tiene como nombre `emp.xsd`, un procesador que cargue este documento podría tratar de manera automática de validarlo referenciado al schema como sigue[195]:

```
<empleado xmlns="http://hr/empleados"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation "http://hr/empleados emp.xsd">

...
```

[193] Ibid., p. 8
[194] Ibid., p. 9
[195] Ibid., p. 10

Esta podría ser una sintaxis difícil de seguir debido a toda la actividad de los namespaces, pero si se observa que el elemento root del documento declara el namespace por default se tiene que es `http://hr/empleados`. Entonces, el namespace de los documentos XMLSchema-instance es declarada y renombrada con el prefijo xsi: El atributo `xsi:schemaLocation` es utilizado para asociar el namespace del documento con la dirección relativa del archivo, provista por `emp.xsd`. Podría también ser posible usar un path absoluto que apunte al schema si no estuviera localizado en el mismo servidor.

Otro método de validación manual de un XML Schema es hacer referencia al mismo sin recurrir a algún alias, apellido, calificativo o namespace. Los documentos que no utilizan namespaces pueden proveer información acerca de la posible localización de un schema de validación. Debido a que no existe namespace, el atributo utilizado es llamado noNamespaceSchemaLocation y también reside en el namespace XML Schema instance de la W3C.

Por ejemplo, si un empleado registrado en el ejemplo anterior no utiliza un namespace, podria referirse a la dirección del schema como sigue:

```
<empleado xsi:noNamespaceSchemaLocation="emp.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

En este caso, el valor del atributo **`xsi:noNamespaceSchemaLocation`** es exclusivamente la ruta al archivo de schema, debido a que no existe namespace al cual se pueda asociar el schema[196].

La validación manual de los XML Schemas, es decir, el anexar una referencia del XML Schema en el documento instancia XML usando la técnica de schemaLocation o noNamespaceSchemaLocation, *eventualmente puede llegar a traer el problema de que muchos interpretes no validan XML por default.* También la recomendación de XML Schemas por la W3C especifica que los procesadores son libres de ignorar estas definiciones de schemas adjuntos. Aunque Microsoft XML Core Services 4 automáticamente valida los documentos XML con los schemas adjuntos, otros problemas pueden ocurrir que limitan la eficacia de este método[197]. Por otra parte, en el mejor de los casos, aunque se le provea el XML al usuario final, no sabría como utilizarlo y aunque se le capacite sobre cómo hacerlo, en algunos casos no se molestaría en hacerlo.

¿Qué otras desventajas se presentan al validar las instancias de XML con los XML Schemas asociados? Por ejemplo, si una compañía recibe órdenes de compra de muchos socios de negocios diferentes, desearía asegurar que cada orden de compra es válida. Sin embargo, no es suficiente para la compañía confiar en diferentes schemas referenciados en cada instancia de documento. En primer lugar, un documento caducado podría referirse a un schema igualmente caducado, pero no ocurrirá un error de validación porque los errores son correspondientes y se genera una correspondencia errónea. Similarmente, problemas de comunicación podrían impedir que un servidor distribuya los schemas desde direcciones remotas, especificadas en la instancia de documento.

Por estas razones, típicamente se aplica un schema a un documento de instancia al **usar métodos de programación sobre cada documento recibido**. Esto sobrescribe cualquier schema del encabezado y permite asegurarse de que se validará siempre contra el schema correcto[198].

Las organizaciones o aplicaciones que desean validar un documento XML contra un schema usualmente no confiarán en el schema anexo, porque este podría referirse a una especificación caducada, corrupta o no recuperable en la red. La única forma de asegurar que un documento es válido con respecto a un schema en particular es usar una copia local de ese esquema a validar. Esto puede hacerse por programa usando un objeto provisto por MSXML4 para el caso de Microsoft, y por JAXP ó MSV de Java 2 SE.

¿Cuáles son los tipos de datos predefinidos en los XML Schemas? Cuando se generan XML Schemas, es importante poder especificar el tipo de contenido que deberían tener los elementos y atributos individuales en los documentos de instancia

[196] Ibid.
[197] Ibid.
[198] Ibid., p. 11

validos. *La recomendación de W3C para los XML Schemas incluye un número de tipos de datos listos para usarse que corresponden a tipos comunes encontrados en lenguajes de programación y sistemas manejadores de base de datos.*

Los tipos de datos predefinidos en los XML Schemas pueden ser primitivos o derivados[199]:

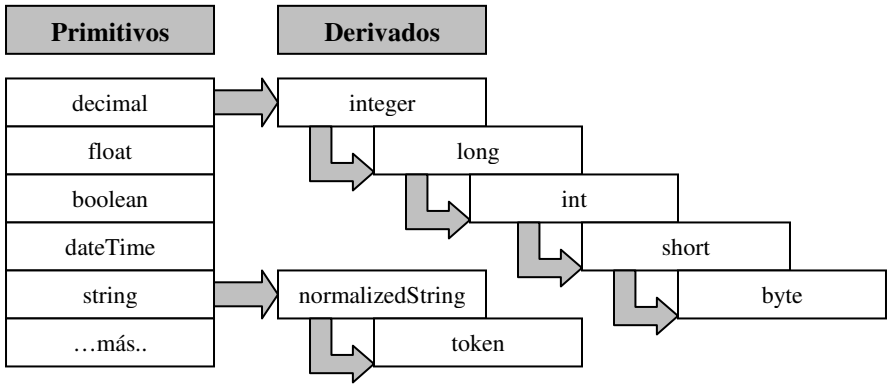


Ilustración 33 Tipos de datos predefinidos en los XML Schemas

¿Qué características reúnen los tipos de datos primitivos en los XML Schemas? Los tipos de datos predefinidos son conjuntos de tipos definidos por la especificación de la W3C sobre XML Schemas. Este conjunto cumple con tipos de datos primitivos que no están definidos en términos de algún otro tipo de datos, y un número de tipos de datos derivados que están basados en los tipos primitivos.

Algunos de los tipos de datos primitivos más comunes están enlistados en la siguiente tabla[200]:

Tipo de Datos	**Descripción**	**Ejemplo**
decimal	Una secuencia de longitud finita de dígitos decimales separados por un indicador decimal.	123.45
float	Un número de punto flotante de 32 bits de precisión simple definido por el Instituto de Ingenieros Eléctricos y Electronicos (IEEE-754-1985)	12.34E-5
boolean	Un valor logico verdadero o falso, representado por los valores “trae”, “false” o “0” y “1”.	true
date	Fecha en el calendario en formato CCYY-MM-DD definido por la Organización Internacional para la Estandarizacion (ISO-8601)	1976-30-08
time	Un instante de tiempo con información opcional acerda de la zona horaria, en el formato hh:mm:ss.sss definido en ISO-8601	17:00:25.66
string	Una secuencia de longitud finita de caracteres definidas por la especificación XML 1.0	_ _ _ _Facultad de Ingeniería

Tabla 12 Tipos de datos primitivos más comunes

En contraparte, ¿Qué especificaciones tienen los tipos de datos derivados en los XML Schemas? Los tipos de datos derivados definidos por la especificación de tipos de datos de la W3C para los XML Schemas están basados tanto en tipos de datos primitivos como en otros tipos de datos derivados. Aunque podría ser posible que esos tipos de datos fueran definidos dentro del schema por el usuario, se decidió que son de uso común por lo que previenen de incluirlos de forma explícita.

Algunos de los tipos de datos derivados más comunes están enlistados en la siguiente tabla[201]:

[199] Ibid., p. 19
[200] Ibid.
[201] Ibid., p. 20

Tipo de Datos	Descripción	Ejemplo
integer	Derivado del tipo de datos primitivo decimal fijando el número de dígitos antes del punto decimal	123
long	Derivado del tipo de datos integer limitando los valores a aquellos que pueden ser representados utilizando la notación binaria de 64 bits	9876543210
int	Derivado del tipo de datos long limitando los valores a aquellos que pueden ser representados utilizando la notación binaria de 32 bits	76543210
short	Derivado del tipo de datos int, limitando los valores a aquellos que pueden ser representados utilizando la notación binaria de 16 bits	54321
byte	Derivados del tipo de datos short, limitando los valores a aquellos que pueden ser representados utilizando la notación binaria de8 bits	21
normalizedstring	Derivado del tipo de datos string, removiendo los caracteres <Intro>,<cr>,<lf> y <tab>	Facultad de Ingeniería
token	Derivado del tipo de datos normalizedString, removiendo los espacios antes y después del texto, así como secuencias internas de dos o más espacios	Facultad de Ingeniería

Tabla 13 Tipos de datos derivados más comunes

Al conocer los tipos de datos se plantea la pregunta: ¿cómo se pueden usar al escribir la declaración de elementos? Cada documento XML debe contener al menos un elemento. Los elementos pueden contener solamente datos o una mezcla de elementos, atributos y datos. Los elementos que contienen solo datos son considerados simples, mientras que los elementos que contienen elementos hijos o que tienen atributos son considerados complejos. Cada tipo de elemento puede ser declarado en un XML Schema usando la estructura **xs:element**[202].

- Los XML Schemas consisten de tipos simples y tipos complejos.
- Los elementos son simples si contienen únicamente datos.
- Los elementos complejos pueden contener elementos y atributos.
- Se pueden crear declaraciones de elemento usando **<xs:element>**

¿Cómo se puede declarar un elemento en un XML Schema? Para que una instancia de documento XML sea considerada válida, cada elemento que aparece en ese documento debe ser declarado en un XML Schema asociado. Se debe declarar el nombre del elemento, y se puede declarar opcionalmente otras propiedades, tales como el tipo de datos que debe contener, sea que deba tener un valor por default o fijo, y el número de veces que el elemento puede ocurrir en un lugar específico.

Para declarar un elemento, se debe crear una instrucción **xs:element**. Si el elemento es declarado como hijo del elemento root de **xs:schema**, se debe dar un nombre y se dice que tiene un alcance global porque puede ser referenciado desde múltiples lugares dentro del Schema. También es posible declarar elementos dentro de **xs:complexType** y **xs:group**, pero dichos elementos no pueden reverenciarse desde algún otro lugar. Estas declaraciones se dice que tienen alcance local.

La sintaxis para declarar un elemento en un XML Schema es como sigue[203]:

```
<xs:element name=”elementname” type=”datatype”
(fixed=”fixedvalue” | default=”defaultvalue”)
minOccurs=”value” maxOccurs=”value” … />
```

¿Qué pasos se dieron para crear el elemento del XML Schema anterior? Son fundamentalmente cuatro[204]:

[202] Ibid., p. 21
[203] Ibid.

1. **Establecer el nombre del elemento**. El nombre de un elemento pordría ser un NCName (non-colonized name) y está definido en la recomendación XML de W3C. Esto es, el nombre de el elemento no puede contener dos puntos, por lo que no se debe incluir un calificador de namespace. Si es requerido, el namespace del elemento debe establecerse usando el atributo targetNamespaceen elelemento xs:schema.
2. **Establecer el tipo del elemento**. El tipo de un elemento puede ser algún tipo de datos predefinido o definido por el usuario. Los tipos de datos definidos por el usuario son creados utilizando las estructuras xs:simpleType o xs:complexType. Las declaraciones de elementos simples siempre se refieren a un tipo de datos preconstruido o a un tipo de datos definido usando la estructura xs:simpleType. Si el tipo de datos es un tipo de datos preconstruido, se debe calificar el nombre del tipo de datos con el namespace del XML schema. Similarmente, si el tipo de datos es uno definido por el usuario, podría tener que estar calificado con el namespace del schema en el cual fue declarado.
3. **Definir un valor por default o fijo**. Se puede usar el atributo **default** para proveer un valor por default que debe inferirse si no se provee contenido para el elemento en el documento de instancia. Simpilarmente, se puede usar el atributo **fixed** para proveer un valor fijo que el elemento debe tener en todos los documentos deisntancia. Estas propiedades son mutuamente excluyentes, por lo que no se puede especificar un valor por default mientras se especifica un valor fijo.
4. **Especificar las mínimas y máximas ocurrencias permitidas**. Si la declaración de elemento es utilizada dentro de otra declaración, por ejemplo una declaración xs:complexType, se puede también especificar el mínimo y máximo número de veces que el elemento es permitido para aparecer, al establecer los valores nuéricos apropiados para los atributos **minOccurs** y **maxOccurs**. Se puede especificar que no existe límite máximo al utilizar la cadena "unbounded" como el valor del atributo **maxOccurs**.

Si los elementos pueden contener atributos dentro de sí, entonces. ¿Cómo pueden declararse tales en un XML Schema? Muchos documentos XML contienen elementos que tienen atributos asociados. Aunque los elementos pueden tener contenido simple o complejo, los atributos siempre son simples, debido a que solo contienen datos de carácter. Los atributos son declarados utilizando la instrucción **xs:attribute**. Las declaraciones de atributo son creadas utilizando la etiqueta **<xs:attribute>**. Los atributos pueden ser declarados globalmente a nivel schema[205].

Por consiguiente, ¿cómo es la metodología de declaración de atributos en un XML Schema? Para que una instancia de documento XML sea considerada válida, cada atributo que aparece en ese documento debe ser declarada en un XML Schema asociado. Se debe declarar el nombre del atributo, y opcionalmente se puede declarar otras propiedades, tales como el tipo de datos que debe contener, sea que deba tener un valor por default o fijo, y sea que el atributo sea opcional o requerido.

Para declarar un atributo, se debe recurrir a la instrucción **xs:attribute**. Si el atributo es declarado como hijo del elemento root **xs:schema**, debe darse un nombre y se dice que tiene alcance global porque puede ser referenciado desde múltiples lugares dentro del schema. También es posible declarar atributos dentro de las estructuras **xs:complexType** y **xs:attributeGroup**, pero dichos atributos no pueden reverenciarse en ningún otro lado. Esas declaraciones son conocidas como de alcance local.

La sintaxis para declarar un atributo en un XML Schema es como sigue[206]:

```
<xs:attribute name="elementname" type="datatype"
(fixed="fixedvalue" | default="defaultvalue")
use=("optional"| "prohibited"| "required")… />
```

¿Cuántas partes existen en la *sintaxis de creación del atributo en el XML Schema* anterior? También son cuatro[207]:

- **Establecer el nombre del atributo**. El nombre de un atributo podría ser un NCName (non-colonized name) y está definido en la recomendación XML de W3C. Esto es, el nombre del elemento no puede contener dos puntos, por lo

[204] Ibid., p. 22
[205] Ibid., p. 23
[206] Ibid.
[207] Ibid., p. 24

que no se debe incluir un calificador de namespace. Si es requerido, el namespace del elemento debe establecerse usando el atributo **targetNamespace** en el elemento **xs:schema**.

- **Establecer el tipo del atributo**. El tipo de un atributo puede ser algún tipo de datos predefinido o definido por el usuario. Los tipos de datos definidos por el usuario son creados utilizando las estructuras **xs:simpleType** o **xs:complexType**. Si el tipo de datos es un tipo de datos preconstruido, se debe calificar el nombre del tipo de datos con el namespace del XML schema. Similarmente, si el tipo de datos es uno definido por el usuario, podría tener que estar calificado con el namespace del schema en el cual fue declarado.
- **Definir un valor por default o fijo**. Se puede usar el atributo **default** para proveer un valor por default que debe inferirse si no se provee contenido para el elemento en el documento de instancia. Simpilarmente, se puede usar el atributo **fixed** para proveer un valor fijo que el atributo debe tener en todos los documentos de instancia. Estas propiedades son mutuamente excluyentes, por lo que no se puede especificar un valor por default mientras se especifica un valor fijo.
- **Especificar si un atributo es requerido**. También es posible especificar si un atributo es obligatorio, estableciendo el atributo **use** en la declaración. Los valores posibles para el atributo **use** son **optional**, **required**, y **prohibited**, en donde **optional** es el default. Si la declaración de atributo es global, especificar que su uso es requerido significa que todas las instancias del atributo a través del documento son requeridas. El valor **prohibited** se utiliza únicamente para restringir la existencia de tipos de datos complejos que tienen una declaración de atributo que no debe aplicar a una dirección en particular.

Aunque las recomendaciones de los XML Schemas de la W3C definen un número de tipos de datos preconstruidos muy útil, siempre existe la necesidad de crear nuevos tipos de datos que se ajustan mucho mejor a un elemento o atributo en particular. Se puede derivar nuevos tipos de datos simples para restringir el contenido de los datos de un atributo o elemento, utilizando la estructura **xs:simpleType**.

Por citar tres ejemplos:

- Se puede crear una restricción para limitar los posibles valores.
- Se puede crear una lista para permitir múltiples valores discretos.
- Se puede crear una unión para combinar tipos de datos existentes.

Ahora bien, ¿Cómo definir el contenido de un simpleType? Se define el contenido de un tipo de datos dentro del elemento en la estructura xs:simpleType. Un tipo de datos simple está siempre basado en otros tipos de datos. Esos tipos de datos pueden ser preconstruidos o definidios por el usuario, y estos pueden ya existir o ser definidos dentro del tipo de datos simple por si mismos. Los tipos de datos simples pueden estar construidos utilizando los nodos hijos de las etiquetas **xs:restriction**, **xs:list** o **xs:union**, como se define en la siguiente tabla[208]:

Estructura	Descripción
xs:restriction	Una restricción establece límites en el conjunto de valores definidos por otro tipo
xs:list	Una lista provee el medio de permitir múltiples valores de otro tipo de datos
xs:union	Una unión permite valores para aparecer en más de algún otro tipo

Tabla 14 Estructuras en los XML Schemas

También, **¿Cómo definir reglas en una restricción del XML Schema?** Para restringir el contenido de un tipo de datos, se debe especificar *que tipo de datos será base para el nuevo tipo derivado*. Entonces se debe especificar una o más reglas o restricciones de restricción. Algunas reglas o circunstancias son listadas en la siguiente tabla[209]:

[208] Ibid., p. 25
[209] Ibid., p. 26

Faceta	Descripción
Lenght	Permite establecer un número fijo de caracteres en una cadena
minLenght	Permite establecer el número mínimo de caracteres en un string
maxLenght	Permite establecer el número máximo de caracteres en un string
minInclusive	Permite establecer el menor valor permitido en un número
maxInclusive	Permite establecer el máximo valor permitido de un número
totalDigits	Permite establecer el número total de dígitos que un número puede tener
fractionDigits	Permite establecer el número de dígitos después del punto decimal
pattern	*Permite especificar una expresión regular con la cual el dato debe cumplir*
ennumeration	Permite proveer una lista de valores permitidos

Tabla 15 Restricciones en los XML Schemas

La restricción de tipo **pattern**, es utilizada para especificar expresiones regulares. Es imprescindible en el filtrado de tipos de datos especiales, ya que *produce un ahorro de tiempo de programación en la compañía para validación de datos*. Por ejemplo, se puede definir un tipo de datos llamado placasVehiculares que establece un patrón en donde existan tres letras mayúsculas, y un guión, y tres dígitos en ese orden. Esta regla codificada en XML Schema toma la siguiente forma:

```
<xs:simpleType name="placasVehiculares">
      <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z]{3}-\d{3}">
      </xs:restriction>
</xs:simpleType>
```

De hecho, en la tesis se hace un extenso uso de pattern, y en la sección 4.4 se hará mención de cómo dicha característica simplifica notablemente la codificación de la aplicación.

La gramática XML Schemas está orientada a objetos, debido a que permite la reutilización de estructuras. En el caso de los tipos de datos simples, se tiene que pueden ser reutilizados Cuando se declara un atributo o elemento en un XML Schema, y se desea especificar que su tipo de dato es uno que se ha derivado utilizando una estructura **xs:simpleType**, se puede hacer anónima o explícita, dependiendo de cuantas veces se espera utilizar.

Se puede declarar un nuevo tipo simple incrustando la instrucción **xs:simpleType** dentro de la declaración de un elemento o atributo. En este caso, el tipo se dice que es anónimo, por lo que no se le da un nombre. Los tipos anónimos no pueden usarse fuera del elemento o atributo en el cual fueron definidos.

Alternativamente, si se declara un nuevo tipo simple como hijo de un elemento root **xs:schema**, se le debe dar un nombre. En este caso, se dice que el tipo es explícito, y puede utilizarse a través de todo el schema[210]:

- Si el tipo de datos simple será utilizado una sola vez, es conveniente que sea anónimo, ya que estos tipos de datos no pueden ser reutilizados.

```
<xs:element name="listaDeClientes">
      <xs:simpleType>
            <xs:list itemType="xs:string">
      </xs:simpleType>
<xs:element>
```

- Si el tipo de datos simple es utilizado dos o más veces, es necesario que sea explícito y dependa del elemento root.

```
<xs:attribute name="viejaPlaca" type="placasVehiculares">
<xs:attribute name="nuevaPlaca" type="placasVehiculares">

<xs:simpleType name="placasVehiculares">
      <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z]{3}-\d{3}">
      </xs:restriction>
```

[210] Ibid., p. 28

```
    </xs:simpleType>
```

No solo existen elementos simples dentro de un documento XML. También existen anidamientos, lo que implica definir tipos de datos complejos en el XML que describe a la gramática de dicho documento XML inicial. Los tipos simples permiten especificar el contenido de datos de elementos y atributos, pero la mayoría de los documentos XML consisten de elementos que contienen combinaciones de otros elementos, atributos y datos. Para representar este contenido, se deben crear tipos de datos complejos. **Se pueden crear diferentes tipos de datos complejos. Un tipo de datos complejo es un elemento que contiene una combinación de otros elementos, atributos, o datos**.

Las estructuras básicas de los tipos de datos complejos se muestran a continuación[211]:

Estructura	Descripción

`<xs:sequence>`
`  <xs:element …/>`
`  <xs:element …/>`
`</xs:sequence>`
	Secuencia de elementos

`<xs:attribute …/>` `<xs:attribute …/>`	*Elementos con atributos*
`<xs:group ref="…"/>` `<xs:attributeGroup ref="…"/>`	*Grupos de elementos o atributos*

`<xs:choice>`
`  <xs:element …/>`
`  <xs:element …/>`
`</xs:choice>`
	Selección de algún elemento

`<xs:all>`
`  <xs:element …/>`
`  <xs:element …/>`
`</xs:all>`
	Los elementos pueden venir en cualquier orden.

Tabla 16 Estructuras básicas de los tipos de datos complejos en los XML Schemas

¿Qué ejemplo puede ilustrar los anidamientos en un XML de instancia, lo que obliga a definir tipos de datos complejos en los XML Schemas? El siguiente elemento XML contiene un número de elementos hijos[212]:

```
<direccion type="shipping">
   <calle>Presidentes 144</calle>
   <delegacion>Magdalena Contreras</delegacion>
   <codigoPostal>01214</codigoPostal >
   <estado>Distrito Federal</estado>
</direccion>
```

Obviamente no es posible representar el contenido del elemento `<direccion>` usando un tipo de datos simple, por lo que se debe crear un tipo de datos que declara también a los elementos hijos `<calle>`, `<delegacion>`, `<codigoPostal>` y `<estado>`.

Debido a las ricas estructuras que pueden tener los documentos XML, se necesitará representar muchos tipos de elementos de diferente contenido. Los tipos de datos complejos representan elementos con cualquier tipo de contenido.

¿Cómo definir tipos de datos complejos en un XML Schema? Se declaran tipos de elementos complejos que contienen elementos hijos utilizando la instrucción **xs:complexType** dentro de la declaración de elementos. Se puede entonces especificar una secuencia de elementos hijos, una selección de elementos hijos, o establecer una lísta desordenada de elementos hijos. Si un tipo de datos complejos es declarado en el nivel root del schema, debe ser explícitamente nombrado,

[211] Ibid., p. 33
[212] Ibid.

pero también se pueden declarar tipos complejos anónimos dentro de las declaraciones de elementos si solamente son requeridas en dicha localidad[213].

A continuación se mostrará una declaración explícita comparada con una declaración anónima:

Declaración de tipo complejo en forma anónima	Declaración de tipo complejo en forma explícita

<xs:element name="book">
  <xs:complexType>
  <xs:sequence>
  <xs:element name="title" type="xs:string">
  <xs:element name="isbn" type="xs:string">
  <xs:element name="publisher" type="xs:string">
  </xs:sequence>
  </xs:complexType>
</xs:element>
	<xs:complexType name="booktype">
	  <xs:sequence>
	  <xs:element name="title" type="xs:string">
	  <xs:element name="isbn" type="xs:string">
	  <xs:element
	  name="publisher"
	  type="xs:string">
	  </xs:sequence>
	</xs:complexType>
	
	<xs:element name="book" type="booktype">

Tabla 17 Declaraciones anónimas y explícitas en los XML Schemas

Para ambos casos, el siguiente documento XML es un ejemplo de una instancia válida del elemento <book>:

```
<book>
  <title>Programming in CORBA</title>
  <isbn>9-001-122-01</isbn>
  <publisher>Michi Henning and Steve Vinosky</publisher>
</book>
```

Utilizando tipos de datos complejos, ¿Qué instrucción de los XML Schemas debe usarse al seleccionar solo un elemento? Si se necesita declarar un elemento que solamente contiene uno de un conjunto predefinido de elementos hijos, se debe utilizar una instrucción xs:complexType con una instrucción insertada xs:choice como sigue[214]:

```
<xs:element name="contactmethod">
  <xs:complexType>
    <xs:choice>
      <xs:element name="telephone" type="xs:string">
      <xs:element name="email" type="xs:string">
      <xs:element name="fax" type="xs:string">
    </xs:choice>
  </xs:complexType>
</xs:element>
```

En este ejemplo, un elemento <contactmethod> es declarado que puede tener solo uno de los elementos hijos declarados, es decir <telephone>, <email> o <fax>. Una instancia de elemento válida que satisface esa declaración podría ser la siguiente:

```
<contactmethod>
  <fax>555-1234</fax>
</contactmethod>
```

En otro caso, ¿Cómo declarar tipos de datos complejos con elementos hijos que aparecen en cualquier orden? Se pueden declarar elementos complejos que contienen elementos hijos opcionales en cualquier orden al crear una instrucción **xs:complexType** e insertando una plantilla **xs:all** como sigue[215]:

```
<xs:element name="contactinfo">
  <xs:complexType>
    <xs:all>
      <xs:element name="telephone" type="xs:string">
      <xs:element name="email" type="xs:string">
      <xs:element name="fax" type="xs:string">
      <xs:element name="smsPager" type="xs:string">
    </xs:all>
  </xs:complexType>
```

[213] Ibid., pp. 35,36
[214] Ibid., p. 36
[215] Ibid., p. 37

```
</xs:element>
```

Aquí, el elemento `<contactInfo>` puede tener una instancia de cada elemento hijo `<telephone>`, `<fax>`, `<email>`, `<smsPager>` y puede aparecer en cualquier orden. Para especificar que alguno de esos elementos hijos son opcionales, se podría colocar un atributo **`minOccurs`** en la declaración de elemento con el valor **`0`**. Es interesante notar que no es posible especificar un atributo **`maxOccurs`** con el valor mayor a **1** en la declaración de elemento dentro de una instrucción **`xs:all`**

Al conocer los elementos básicos de los XML Schemas, cabe preguntarse, ¿cómo se realiza la conversión de elementos simpleType a complexType con atributos? En muchos XML Schemas, se necesita crear declaraciones para elementos que contienen atributos. Sin considerar el contenido del elemento, algún elemento que tenga uno o más atributos es considerado un tipo de datos complejo y debe ser declarado usando la instrucción **xs:complexType**. Los atributos pueden añadirse tanto en elementos simples o complejos, usando la instrucción **xs:attribute**

A continuación se muestran los XML de instancia y el schema que describe la gramática de tales documentos XML[216]. Se debe poner especial cuidado en el uso de la instrucción **xs:attribute**:

XML de Instancia	XML Schema que describe su gramática

`<empleado`
`email="vigmms@hotmail.com"`
`departamento="desarrollo"``>`
`  <nombre>Saul Moreno</nombre>`
`  <titulo>Proyect Leader</titulo>`
`  <salario>25300</salario>`
`</empleado>`
	`<xs:element name="empleado">`
	`  <xs:complexType>`
	`  <xs:sequence>`
	`  <xs:element name="nombre" type="xs:string">`
	`  <xs:element name="titulo" type="xs:string">`
	`  <xs:element name="salario" type="xs:integer">`
	`  </xs:sequence>`
	`  <xs:attribute name="email" type="xs:string">`
	`  <xs:attribute name="departamento" type="xs:string">`
	`  </xs:complexType>`
	`</xs:element>`

`<Longitud` **`Unidades="Metros"`**`>12.75</Longitud>`	`<xs:element name="Longitud">`
	`  <xs:complexType>`
	`  <xs:simpleContent>`
	`  <xs:extension base="xs:decimal">`
	`  <xs:attribute name="Unidades" type="xs:string"/>`
	`  </xs:extension>`
	`  </xs:simpleContent>`
	`  </xs:complexType>`
	`</xs:element>`

Tabla 18 XML de Instancia VS XML Schemas de su gramática

En los tipos de datos complejos, **¿es posible hacer referencia a declaraciones existentes para utilizar estructuras dentro del XML Schema?** Sí. De hecho, es frecuentemente útil poder declarar elementos y atributos en un lugar y usarlos múltiples veces. Elementos y atributos pueden ser declarados globalmente en el nivel schema, y esas declaraciones globales pueden referenciarse dentro de complexType y group[217].

Por ejemplo, el primer paso es declarar elementos y atributos globales en el nivel del elemento root <schema>

```
<xs:schema>
 <xs:element name="titulo" type="xs:string"/>
 <xs:attribute name="email" type="xs:string"/>
</xs;schema>
```

Despues, se puede hacer referencia a estas declaraciones de elementos y atributos globales dentro de tipos de datos complejos y grupos.

```
<xs:element name="empleado">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="nombre" type="xs:string"/>
```

[216] Ibid. p. 38
[217] Ibid. p. 41

```
    <xs:element ref="titulo"/>
   </xs:sequence>
   <xs:attribute ref="email" use="required"/>
  </xs:complexType>
</xs:element>
```

Cuando se declara un elemento o atributo, se puede escoger entre declararlo como hijo de la instrucción **xs:schema** o crearla declaración en la dirección en donde el elemento o atributo ocurre, por ejemplo, dentro de la construcción **xs:complexType**.

Si la declaración es creada en el nivel schema, el nombre del atributo debe proveerse y el elemento o atributo debe tener un alcance global. Si la declaración es creada dentro de otro tipo, el nombre del atributo todavía debe proveerse, pero el elemento o atributo tendrá solamente alcance local. Si se debe referenciar un elemento o atributo en lugares diferentes dentro del mismo XML schema, debe crearse sus declaraciones con alcance global.

Para referenciar un elemento o atributo global, se debe usar el atributo **ref** en la instrucción **xs:element** o **xs:atribute** para referirse a la declaración apropiada. El valor del atributo **ref** debe corresponder al **name** del atributo en una declaración global.

Se han citado las características clave de los XML que se utilizan en la presente tesis. Sin embargo, el tema de los XML Schemas es muy amplio y mencionar cada detalle de dicha tecnología está fuera de los objetivos del presente trabajo de investigación, pero si se desea conocer con todo detalle este tema, también puede recurrirse a el Web site D VInt Productions, (http://www.xml.dvint.com) y a las presentaciones xml-schemas1.ppt y xml-schemas2.ppt de Roger L. Costello en XML Technologies Course. Tales son excelentes recursos docentes que dan una magnífica panorámica general de los XML Schemas.

3.2.2 SOAP: Uso de XML para intercambio de información

Puesto que la tecnología de Web services ha sido seleccionada, es imprescindible profundizar sobre los bloques de construcción de un servicio de software estilo Web. En el bloque de formato de mensaje con SOAP, no basta con saber que éste protocolo sirve para intercambiar datos entre el cliente y el servidor, o que SOAP es un acuerdo común de codificar y formatear los mensajes –lo cual es muy necesario, ya que una forma estandarizada de codificar los datos asegura que los mensajes codificados por el cliente deben ser interpretados apropiadamente por el servidor-. Se requiere saber también que, sin una forma estándar de representar los mensajes, desarrollar una herramienta para alejar al desarrollador de los protocolos subyacentes es muy difícil, pero al crear una capa de abstracción entre el desarrollador y los protocolos, se permite que el desarrollador se concentre más en el problema de negocios, y menos en la infraestructura requerida para implementar la solución.

Pues bien, dentro de esta capa de abstracción se pueden citar un par de categorías de mensajes SOAP: los mensajes orientados a procedimiento y los orientados a documento. Los mensajes orientados a procedimiento proveen comunicación de dos vías y son comúnmente referidos como mensajes de llamada a procedimientos remotos (RPC). El cuerpo de un mensaje RPC contiene información acerca de la acción solicitada por el servidor, así como algunos parámetros de entrada/salida. Los mensajes orientados a documento generalmente facilitan la comunicación de una vía. Los documentos de negocios tales como órdenes de compra son ejemplos de mensajes orientados a documento.[218]

¿Qué uso se le puede dar a estos tipos de mensajes? Dos mensajes SOAP son asociados para facilitar las llamadas a métodos RPC con SOAP: el mensaje de solicitud y su correspondiente mensaje de respuesta. La información acerca del método destino con algunos parámetros de entrada es enviada al servidor en el SOAP request. El servidor entonces invoca alguna conducta en representación del cliente y regresa los resultados, así como algunos parámetros de retorno.

Un documento de negocios tal como una orden de compra o una factura puede ser codificado dentro del cuerpo del mensaje SOAP, y direccionado a su recipiente final. El recipiente del documento podría o podría no enviar un mensaje de confirmación de regreso al remitente. Debido a que los documentos de negocios usualmente son enviados a través de

[218] SHORT, Scott. Op. Cit., p.36

múltiples compañias, las organizaciones tales como BizTalk.org y RosettaNet sirven como proveedores y repositorios para esquemas que definen intercambios de documentos comunes[219].

¿Cómo es un mensaje SOAP? ¿Cómo se vería al guardarlo en disco? SOAP provee una manera estándar de empacar un mensaje. Un mensaje SOAP está compuesto de un Envelope que contiene el cuerpo del mensaje y alguna información para el encabezado utilizada para describir el mensaje. Aquí se muestra un ejemplo:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Optional header information goes here. -->
    <To>Scott</To>
    <From>Suzanne</From>
  </soap:Header>

  <soap:Body>
    <!--Message goes here. -->
    Please pick up some milk on your way home from work.
  </soap:Body>
</soap:Envelope>
```

Ilustración 34 Ejemplo de un mensaje SOAP

- **Envelope**. El elemento root del documento es el elemento Envelope. Este ejemplo contiene dos subelementos, Body y Header. Un mensaje SOAP válido puede también contener otros elementos hijos dentro del sobre.
- **Header**. El envelope puede contener un elemento Header opcional, el cual contiene información acerca del mensaje. En el ejemplo anterior, el header contiene dos elementos describiendo al individuo que compone el mensaje y el recipiente destino del mensaje.
- **Body**. El envelope debe contener un elemento Body. El body contiene la carga de datos central del mensaje[220].

¿Qué función tiene cada uno de los elementos en ésta estructura? La primera parte del SOAP request es un encabezado estándar HTTP que indica que la petición es una operación del tipo HTTP POST, cuyo Universal Resource Identifier (URI) es usualmente definido en la primer línea. El campo Content-Type muestra que la carga de datos en http es un documento XML, y el campo SOAPAction le dice al host remoto que el contenido es un mensaje SOAP. *SOAPAction es casi siempre asignado al nombre del método para invocar*, por lo que el web Server o el firewall deben realizar este filtro de mensajes de alto nivel.

La segunda parte del SOAP request es el documento XML que consiste de tres porciones principales:

- **Envelope.** El envelope define los diversos namespaces de XML que son utilizados por el resto del mensaje SOAP, y típicamente incluyen a xmlns:soap (SOAP envelope namespace), xmlns:xsi (XML Schema for instances) xmlns:xsd (XML Schema for data types) y tambien xmlns:soapenc (SOAP encoding namespace).
- **Header.** El encabezado es un elemento opcional para transferir información auxiliar para identificación en seguridad, transacciones, ruteo y pagos de datos. Algún elemento en una cadena de procesamiento SOAP puede añadir o borrar objetos del encabezado; los elementos pueden optar también por ignorar los objetos desconocidos. Si un encabezado está presente, debe ser el primer hijo en el envelope.
- **Body.** El cuerpo es la principal carga de datos del mensaje. Cuando SOAP es utilizado para realizar una llmadaRPC, el cuerpo contiene un elemento sencillo que contiene el nombre del método y sus argumentos.El namespace del nombre del método es especificado por el web service, y en los casos generales este es igual a http://tempuri.org/ segido por el tipo de web service destino. El tipo de cada argumento puede ser opcionalmente

[219] Ibid., p.37
[220] Ibid., pp. 31, 32

> reemplazado usando el atributo xsi:type […]Si un header está presente, el cuerpo debe seguirle inmediatamente; de otra forma debe ser el primer hijo del envelope.[221]

¿Qué ocurre cuando el servicio Web necesita notificar al cliente de Web acerca de un error? Para tal caso se tiene a los SOAPExceptions que son casos particulares de SOAPResponses. Entonces, si ocurre una excepción en algún momento durante el procesamiento de un mensaje, una falla SOAP es generada y codificada en una forma similar a una respuesta SOAP regular. El encabezado de respuesta HTTP estándar indica que la respuesta se trata de una excepción[222]. La carga de datos de XML contiene un envelope y un body tal como una respuesta regular, excepto que el contenido del cuerpo es una estructura soap:Fault cuyos campos son definidos como siguen:

- **faultcode**. Un código que indica el tipo de falla.Los valores válidos son soap:Client (el mensaje se formó incorrectamente), soap:Server (problema de envío), soap:Version;ismatch (namespace invalido para el elemento Envelope) y soap:MustUnderstand (error en el procesamiento del contenido del encabezado)
- **faultstring**. Una descripción legible por el ser humano acerca de la falla.
- **faultactor.** Un campo opcional que indica el URL de la fuente de falla.
- **detail.** Un documento XML específico de la aplicación que contiene información detallada acerca de la falla[223].

Es interesante el hecho de que el faultstring regrese una descripción en el mismo idioma del operador del sistema, puesto que eso permite que tanto los usuarios como los programas basen su toma de decisiones a partir de ese mensaje.

Algunas implementaciones SOAP añaden un elemento adicional para codificar información acerca de las excepciones remotas tales como su tipo, dato y volcado de pila, información que puede ser utilizada por el cliente para tomar una acción automática.

¿De qué manera puede SOAP ser utilizado para efectuar llamadas de RPC en objetos distribuidos? Una de sus metas de diseño originales fue proveer una forma abierta y estándar para facilitar RPCs usando las tecnologías de Internet tales como XML y HTTP. Además, una gran ventaja de la especificación SOAP en este sentido es que no dicta la forma en que el mensaje debe ser codificado. Entonces, a pesar que se puede recomendar una forma de codificar los mensajes de request y response, el desarrollador está libre para crear su propio método de codificar las comunicaciones RPC[224].

Ahora bien, el concepto de RPC –que es utilizar una función remota, como si fuera local, utilizando un conjunto finito de parámetros- , nos hace recordar que en la programación estructurada las funciones no solamente aceptan parámetros por valor, sino también por referencia. ¿Qué ocurre en el caso de los servicios de software en web? Hasta este punto, se ha venido explicando como codificar los parámetros que son pasados por valor a los Web services. Pero es usualmente necesario pasar los parámetros por referencia. Por ejemplo, un cliente podría pasar información acerca de un cliente al servidor, por lo que el servidor podría actualizar la información en representación del cliente. Si la estructura del cliente fuera pasada por valor, los cambios hechos a la información del cliente podrían no ser visibles al cliente.[225]

Otra razón para pasar los parámetros por referencia es llanamente hablando, para mantener la identidad de la variable utilizada por el Cliente[226].

¿Porqué se debe usar una especificación de XML como SOAP, y no simplemente dejar que cada quien envíe los mensajes como desee definirlos? Esto se debe a que SOAP es una gramática particular de XML para enviar los datos con descriptores. Usualmente es necesario incluir metadata adicional con el cuerpo del mensaje. Por ejemplo, quizás se necesite incluir información acerca de los tipos de servicios que el Web service necesita para llenar correctamente la requisición, tal

---

[221] GLASS. Op. Cit., p. 14
[222] Para más información sobre el encabezado de HTTP Vid. supra, 3.1 Protocolo HTTP
[223] GLASS. Op. Cit., pp. 15, 16
[224] SHORT, Scott. Op. Cit., pp. 38,39
[225] Ibid., p. 49
[226] Ibid., p. 50

como listar una transacción o redireccionar la información a otro servidor. El XML puro no provee un mecanismo para diferenciar el cuerpo del mensaje de sus datos asociados.

Aquí, Simple Object Access Protocol (SOAP) provee un medio de asociar la información de encabezado de un mensaje con el cuerpo del mismo mensaje, sin necesidad de cambiar el protocolo de transporte. Cada mensaje de SOAP debe definir un Envelope. El Envelope tiene un cuerpo que contiene la carga de datos del mensaje y el header que puede contener metadata asociada con el mensaje[227].

¿Qué otras razones hay, por las cuales SOAP ha sobresalido entre otros protocolos? Además de ser SOAP la piedra angular de los Web services –gracias a que delinea una forma estándar de empaquetar los mensajes-, SOAP ha recibido mucha atención debido a que facilita las comunicaciones en el estilo RPC entre el cliente y algún servidor remoto. A pesar que muchos protocolos han sido creados para facilitar las comunicaciones entre dos aplicaciones –incluyendo RPC de Sun, DCE de Microsoft, RMI de Java, y ORPC de CORBA- SOAP ha recibido mucha atención gracias a que tiene un soporte de la industria sin precedente. SOAP es el primer protocolo en su tipo que ha sido aceptado por prácticamente cada compañía grande de software en el mundo. Las compañías que raramente cooperan entre sí están compitiendo alrededor de este protocolo. Algunas de las grandes compañías que están soportando a SOAP son Microsoft, IBM, Sun Microsystems, SAP y Ariba[228].

Otras grandes ventajas que debe mencionarse en este punto son las siguientes:

- **No está fuertemente asociado a algún lenguaje**. Los desarrolladores envueltos en nuevos proyectos pueden escoger desarrollar en el lenguaje de programación más reciente y más grandioso. Pero los desarrolladores que son responsables de mantener aplicaciones de largo plazo podrían no tener alguna selección específica del lenguaje de programación que ellos utilizarán. SOAP no especifica una API, por lo que la implementación de la API se realiza por el lenguaje de programación (tal como Java) y la plataforma (como Microsoft .NET).
- **No está fuertemente asociado a un protocolo de transporte en particular**. La especificación SOAP describe cómo los mensajes SOAP deben ser adaptados para http. Pero un mensaje SOAP no es más que un documento XML, por lo que puede ser transportado sobre cualquier protocolo que tenga la capacidad de transmitir texto.
- **No se encuentra amarrado a alguna infraestructura de objetos distribuidos**. La mayoría de los sistemas de objetos distribuidos pueden ser extendidas (y algunas de ellas lo son) para soportar SOAP. Es importante notar que aún con SOAP, el middleware tal como COM+ todavía juega un papel importante en la empresa. El middleware de componentes es todavía responsable de algunas de las características mas avanzadas de la administración de objetos, tal como el manejo del tiempo de vida del objeto, transacciones, creación de un fondo común para todos los objetos, y concentrado de los recursos en un repositorio específico. SOAP habilita un grado de interoperabilidad entre diferentes sistemas que están ejecutando middleware de componentes de vendedores competitivos.
- **Descansa en los estándares de la industria existentes.** Los contribuidores principales a la especificación SOAP intencionalmente evitaron reinventar cualquier cosa. Optaron por crear una extensión de los estándares existentes para alcanzar sus necesidades. Por ejemplo, SOAP se apoya en XML para codificar los mensajes. En lugar de usar su propio sistema de tipos de datos, SOAP utiliza las definiciones de tipos ya definidas dentro de la especificación de los esquemas de XML. […] SOAP no define los medios de transportar los mensajes; SOAP puede estar asociado a los protocolos de transporte existentes, tales como HTTP y SMTP.
- **Habilita la interoperabilidad a través de múltiples ambientes.** SOAP fue construido sobre los estándares de la industria ya existentes, por lo que las aplicaciones corriendo en plataformas que soportan esos estándares pueden comunicarse efectivamente a través de los mensajes de SOAP con aplicaciones corriendo en otras plataformas. Por ejemplo, una aplicación de escritorio corriendo en una PC puede comunicarse de manera efectiva con una aplicación final corriendo en un mainframe que es capaz de enviar y recibir XML sobre http[229].

Ahora bien, ¿qué especificación de SOAP debe utilizarse? La recomendación oficial por la W3C es SOAP versión 1.1. La mayoría de los productos Microsoft –y Sun Microsystems- que incorporan a SOAP no adoptarán SOAP 1.2 hasta que se

[227] Ibid., p. 7
[228] Ibid., p. 29
[229] Ibid., pp. 29, 30

convierta la recomendación oficial de W3C. Se recomienda aprender el protocolo SOAP 1.1 con un ojo en los deltas de la versión 1.2 [230]. De hecho, en la programación de la presente tesis, solamente utilizamos SOAP v 1.1.

SOAP versión 1.1. maneja un concepto de alta versatilidad en los sistemas de objetos distribuidos. Tal concepto son los actores. El viejo esquema cliente/servidor es superado con la idea de que un mismo mensaje puede ser procesado por un conjunto de servicios Web. Un SOAP Actor es en este sentido, algo que actúa en el contenido de un mensaje SOAP. Existen dos tipos de SOAP Actors, "default actors" e "intermediarios".

El default actor es el recipiente final proyectado para un mensaje SOAP. Un intermediario recibe un mensaje SOAP y podría actuar sobre el mensaje (incluyendo o modificándolo de alguna forma) antes de reenviarlo a través de la ruta del mensaje planeada, como se muestra en la figura. Aún cuando los intermediarios podrían modificar los datos transferidos del cliente al default actor, todavía se considera como el mismo mensaje[231].

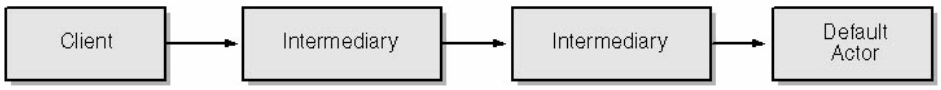


Ilustración 35 Un mensaje SOAP puede ser procesado por varios intermediarios antes de alcanzar su destino

Aquí es donde se contesta la pregunta, ¿Qué usos tiene el SOAP Header? ¿No existe ya un HTTP Header? El elemento opcional Header es utilizado para pasar datos que podría no ser apropiado codificarlos en el cuerpo del mensaje. Por ejemplo, si el default actor recibe un mensaje en el cual es body está comprimido, el default actor podría necesitar conocer que tipo de algoritmo de compresión fue utilizado, a fin de descomprimir el mensaje. Incrustar información acerca del algoritmo de compresión en el cuerpo del mensaje no tiene sentido debido a que el cuerpo por si mismo está comprimido. Colocar este tipo de información en el header del mensaje es más apropiado[232]. En cambio, el HTTP Header solo contiene el SOAP Action, que es una cadena que le dice al servidor qué método remoto se está ejecutando vía RPC o por documento.

El SOAP Header es muy versátil. Otras formas en que puede ser utilizado se detallan a continuación:

- **Identificación de seguridad**. El recipiente podría necesitar que el servidor se identificara por sí mismo antes de que el mensaje pueda ser procesado.
- **Información adicional de seguridad**. Si el recipiente necesita asegurarse de que el contenido del mensaje no ha sido alterado, el remitente puede firmar digitalmente el cuerpo del mensaje y colocar el resumen de esa información en el header.
- **Información de ruta**. Si el mensaje necesita enviarse a varios destinos, los destinos así como su orden pueden ser incluidos en el encabezado.
- **Transacciones**. El recipiente podría tener que ejecutar alguna acción en el ámbito de las transacciones del remitente.
- **Información sobre facturación**. Si el recipiente del mensaje provee servicios al cliente basado en una tarifa por uso, la información necesaria para recolectar los pagos puede estar incrustada en el header[233].

Ya se ha hablado mucho del SOAP Header. ¿Cómo debe utilizarse? Scott SHORT[234] dice que el elemento Header puede añadirse como un elemento hijo inmediato dentro del SOAP Envelope. La información del header aparece como nodos hijos dentro del elemento SOAP Header. Aquí se muestra el ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Digest>B839D234A3F87</Digest>
```

[230] Ibid., p. 31
[231] Ibid., p. 32
[232] Ibid.
[233] Ibid., pp. 32, 33
[234] Ibid., p. 33

```
  </soap:Header>
  <soap:Body>
    <StockReport>
      <Symbol>MSFT</Symbol>
      <Price>74.56</Price>
    </StockReport>
  </soap:Body>
</soap:Envelope>
```

El mensaje SOAP contiene un elemento Digest en el header que la aplicación remota puede utilizar para asegurarse que el mensaje no ha sido alterado. Si el cliente está ejecutando una revisión de rutina para obtener algún dato del inventario, podría no importarle validar su mensaje. **Pero si el precio del inventario dispara un evento dentro del paquete de software financiero, entonces debe estar más interesado en validar el mensaje**. Por ejemplo, podría ser muy desafortunado si el paquete de sofware financiero liquidara automáticamente su portafolio, como resultado de recibir un mensaje de prueba enviado por algún niño de 14 años.

Ahora bien, no siempre se tendrán más de dos actores en el procesamiento del mensaje. Habrá Web services que manejen el esquema simplificado de Cliente/Default Actor. Para este esquema mínimo quizás el SOAP Header no sea tan indispensable, ni su utilización tan crítica. ¿Cómo entonces, podemos diferenciar un SOAP Header informativo contra un SOAP Header crítico?

Debido a que los encabezados son opcionales, el recipiente del mensaje puede optar por ignorarlo. Sin embargo, cierta información que puede ser incrustada en el SOAP Header no debe ser ignorada por el recipiente destino. Si el header no es comprendido o no puede ser manejado adecuadamente, la aplicación podría no funcionar apropiadamente. Por tanto, se requiere una forma de distinguir entre la información del header que es informativa y la información del header que es crítica.

Se puede especificar si el recipiente del mensaje debe comprender un elemento en el header especificando el atributo mustUnderstand con un valor de 1, en la raíz del elemento Header. Por ejemplo, el mensaje SOAP podría solicitar que una aplicación remota realice una acción en representación del cliente. El siguiente ejemplo actualiza información de la cuenta de usuario dentro del dominio de una transacción:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <TransactionId soap:mustUnderstand="1">123</TransactionId>
  </soap:Header>
  <soap:Body>
    <UpdateAccountInfo>
      <email>sshort@microsoft.com</email>
      <firstName>Scott</firstName>
      <lastName>Short</lastName>
    </UpdateAccountInfo>
  </soap:Body>
</soap:Envelope>
```

El recipiente del mensaje debe actualizar la información de la cuenta del usuario dentro del alcance de la transacción del cliente. Si la transacción es abortada, la aplicación remota debe cancelar los cambios solicitados a la información de la cuenta de usuario. Por tanto, en el ejemplo se codifica la identificación de la transacción dentro del header y se establece el atributo mustUnderstand a 1. La aplicación remota debe llevar a cabo la transacción o sencillamente no procesar el mensaje[235].

[235] Ibid., pp. 33, 34

¿Qué otra regla debe seguirse, además del atributo mustUnderstand, para que el mensaje SOAP sea transferido de punto a punto o a través de varios intermediarios antes de que alcance su destino final? En el caso de que un mensaje SOAP sea dirigido a través de muchos intermediarios antes de alcanzar su destino final, podría necesitarse especificar claramente que el header está diseñado para procesarse por los intermediarios de la transacción, en lugar del default actor.

La especificación SOAP provee el atributo actor para anotar los SOAP Headers diseñados para ciertos intermediarios. El valor de este atributo es el Uniform Resource Identifier (URI) del intermediario por el cual la porción del mensaje será procesada. Si el header está diseñado para ser procesado por el siguiente intermediario para recibir el mensaje SOAP, el atributo actor puede establecerse a http://schemas.xmlsoap.org/soap/actor/next. De otra forma, el atributo actor puede establecerse a un URI que identifique a un intermediario en específico[236].

He aquí un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <TransactionId soap:mustUnderstand="1"
      actor="urn:TransactionCoordinator>123</TransactionId>
  </soap:Header>
  <soap:Body>
    <TransferFunds>
      <Source>804039836</Source>
      <Destination>804039836</Destination>
      <Amount>151.43</Amount>
    </TransferFunds >
  </soap:Body>
</soap:Envelope>
```

Debido a que el elemento en el encabezado TransactionId ha sido diseñado para el intermediario que coordina las transacciones, su atributo actor se establece como el URI del intermediario. El atributo mustUnderstand ha sido también creado, por lo que si el intermediario que coordina las transacciones no comprende el elemento del header TransactionID, debe generar un error[237].

¿Qué ocurre con el SOAP Header cuando es recibido por un intermediario? Si el mensaje es transferido a otro recipiente, algunos elementos header diseñados para el intermediario deben ser removidos antes de que el mensaje sea reenviado. El intermediario puede, sin embargo, añadir elementos adicionales al header antes de reenviar el mensaje al siguiente recipiente, el intermediario que coordina las transacciones debe remover la información de ruta antes de enviarlo a la aplicación.

Uno de los puntos importantes a destacar es que enviar el mensaje directamente al default actor no es considerado como un error. Estableciendo el atributo mustUnderstand a 1 en combinación con establecer el atributo Actor a "urn:TransactionCoordinator" no asegura que el mensaje sea encauzado a través del intermediario. Esto solamente significa que si el mensaje alcanza al intermediario que coordina las transacciones, el debe comprender el nodo TransactionId en el header, o bien lanzar un error[238].

Se ha discutido a fondo el concepto de Actores en el escenario de los Web services, pero ¿Porqué alguien desearía incluir más de dos actores en el procesamiento de un simple mensaje? Se requieren intermediarios en los sistemas distribuidos porque cada objeto distribuido lleva a cabo una tarea específica que apoya los objetivos del sistema. Es decir, en algunos casos, el intermediario necesita llevar a cabo una tarea crítica antes de que el mensaje sea reenviado al default actor. Si el mensaje alcanza al intermediario que coordina las transacciones, este debe remover el encabezado TransactionID antes de volver a colocar el mensaje en su camino. Por tanto, el default actor puede revisar si el header TransactionID existe, el cual

[236] Ibid., p. 34
[237] Ibid., pp. 34, 35
[238] Ibid., p. 35

podría indicar que el mensaje no fue procesado a través de los intermediarios apropiados. Sin embargo, determinar si todos los encabezados fueron procesados después que el mensaje alcanzó al default actor no es siempre ideal[239].

Utilizar muchos actores a veces es indispensable aunque no siempre sea conveniente. ¡A veces tiene sus riesgos! Un problema es que no hay forma de saber si un mensaje viajó a través de todos los intermediarios obligatorios, en el orden apropiado. ¿Qué pasaría si un Web service bancario descubre que un mensaje nunca fue encauzado a través del intermediario que administra las transacciones? Si un error ocurriera durante una transferencia de fondos, podría no ser posible deshacer el trabajo realizado por el intermediario en la ruta. Peor aún, el mensaje SOAP podría ser dirigido a través del intermediario de ruteo antes de ser dirigido a través del coordinador de transacciones. Si este es el caso, no hay forma de decir que la aplicación ejecutó su trabajo fuera del alcance de la transacción. Desafortunadamente, SOAP no provee algún mecanismo para asegurar que el mensaje viaja a través de todos los intermediarios en el orden apropiado pero no hay razón para alarmarse ni despreciar la arquitectura, ya que han emergido protocolos para tratar con este problema[240].

Al haber concluido nuestro análisis del elemento SOAP Header, podemos pasar al del SOAP Body. ¿Qué usos tiene dicho elemento? Un mensaje SOAP válido debe tener un elemento Body. El Body contiene la carga de datos central del mensaje. No hay restricciones en cuanto a como debe ser codificado el Body. El mensaje puede ser una sencilla cadena de caracteres, un arreglo de bytes codificado, o XML. El único requerimiento es que el contenido no tenga algunos caracteres que puedan invalidar el documento XML resultante.

La especificación SOAP describe un método de codificación que puede ser utilizado para dar tratamiento serial a los datos en el cuerpo del mensaje. Es una buena idea cumplir con un esquema de codificación establecido como este porque este permite al remitente ínteroperar más fácilmente con el recipiente usando un conjunto bien conocido de reglas de serialización[241]

¿Por qué se necesitan mensajes del tipo SOAP Fault? Porque no todo se realiza siempre como se planeó. Algunas veces, el servidor encontrará un error mientras procesa el mensaje del cliente. SOAP provee una forma estándar de comunicar mensajes de error de regreso al cliente.

Sin considerar que estilo de codificación fue utilizado para crear el mensaje, la especificación SOAP define el formato de un reporte de errores. El cuerpo del mensaje debe contener un elemento Fault con la siguiente estructura[242]:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <soap:faultcode>Client.Security</soap:faultcode>
      <soap:faultstring>Access denied.</soap:faultstring>
      <soap:faultactor>http://abc.com</soap:faultactor>
      <soap:detail>
        <MyError>
          <Originator>File System</Originator>
          <Resource>MySecureFile.txt</Resource>
        </MyError>
      </soap:detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

[239] Ibid.
[240] Ibid., p. 36
[241] Ibid.
[242] Ibid., p. 37

Llama la atención la etiqueta "faultcode". En ella se encuentran dos palabras separadas por un punto. ¿Qué significado tiene este código de falla? El código de falla contiene un valor que es utilizado para determinar por programación la naturaleza del error. La especificación SOAP define un conjunto de códigos de error que se pueden utilizar para describir errores SOAP básicos. Los códigos de falla son listados en la siguiente tabla[243]:

Fault Code	Description
VersionMismatch	Ha sido especificado un namespace invalido para el elemento SOAP envelope.
MustUnderstand	Un elemento hijo inmediato dentro del SOAP header conteniendo un atributo mustUnderstand establecido en 1 no fue comprendido, o bien no fue obedecido por el servidor..
Client	El contenido del mensaje fué encontrado como la principal causa del error. Posibles causas principales de error resultan en un código de falla del cliente, incluyendo un mensaje malformado, o información incompleta dentro del mensaje.
Server	La principal causa del error no fue directamente atribuible al contenido del mensaje. Ejemplos de estos errores resultan en un código de falla del servidor incluyendo que el servidor no pueda obtener los recursos apropiados, como una conexión de base de datos, para procesar el mensaje, o un error lógico durante el procesamiento del mensaje.

Tabla 19 Códigos base para SOAP Fault

Estos códigos de falla no son obligatorios, ni son aún una convención totalmente aceptada al grado de convertirse un estándar, pero es una buena guía para tratar con los errores.

Además de la anterior, existen otras convenciones adicionales en el elemento SOAP faultcode. Se pueden anexar códigos de falla más específicos a los códigos de falla principales de SOAP, usando la notación "dot" y ordenando los códigos de falla individuales desde el menos específico al más específico. Por ejemplo, si el servidor no puede abrir una conexión a base de datos que es requerida para procesar el mensaje del cliente, el siguiente código de falla podría generarse:

```
<faultcode>Server.Database.Connection</faultcode>
```

Debido a que el error no fue el resultado directo del mensaje del cliente, el código de falla base es Server. Un código de falla más descriptivo se anexa al final del código de falla base.[244]

En este sentido, ¿Para qué podrían ser utilizados los otros elementos del SOAP Fault? El elemento faultstring debe contener una cadena legible por el humano que describa el error encontrado. Aquí hay un valor de faultstring para el error conectándose a la base de datos:

```
<faultstring>Unable to open connection to the database.</faultstring>
```

Se puede utilizar el elemento faultactor para indicar la fuente exacta del error. La única excepción es si un intermediario genera el error. Si el error fue generado en algún otro punto diferente al recipiente final del mensaje SOAP, el elemento faultactor debe contener un URI que identifica la fuente del error. De otra forma, el URI puede omitirse[245].

Se han visto muchos aspectos técnicos sobre el sistema de mensajes de los Web services con el protocolo SOAP. Ahora bien, ¿Qué ejemplo práctico se puede dar para mostrar como convertir una función en lenguaje estructurado en un sistema de mensajes Web? A continuación se muestra la generación de un SOAP Request de ejemplo a partir de una función en C Sharp de Microsoft[246]:

Para facilitar el comportamiento request/response solicitado por RPC, se necesitan dos mensajes SOAP: uno para la petición y otro para la respuesta. Aquí se muestra cómo el mensaje de petición podría codificarse para una función C# simple que sume dos números:

[243] Ibid.
[244] Ibid., p. 38
[245] Ibid.
[246] Ibid., p. 39

```
public int Add(int x, int y)
{
    return x + y;
}
```

El método Add acepta dos enteros como parámetros de entrada y pasa el resultado de regreso al cliente como un parámetro de retorno. Los parámetros de entrada deben enpacarse dentro del cuerpo del mensaje de petición por lo que pueden ser enviados a la aplicación objetivo. Esto es llevado a cabo al empacar los parámetros en un formato parecido a una estructura. A continuación se muestra el mensaje de solicitudo para invocar al método Add(1,2):

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add>
      <x>1</x>
      <y>2</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

Al observar detenidamente el sobre de SOAP anterior, es necesario reconocer que se necesita una forma de validar el contenido del SOAP Request para verificar el número y tipo de parámetros. ¿Qué debe hacer el programa si en vez de encontrar a x e y, encuentra a, b y c? ¿Cómo debe procesar el servicio la información si en vez de encontrar a x e y encuentra a y e x, en ese orden? ¿Qué pasará en el caso de que en vez de encontrar el nodo "Add" encontrara el nodo "Suma", usando los mismos parámetros?

Al seguir el análisis, nos damos cuenta de que el elemento Body contiene el elemento Add. Cada uno de los parámetros de entrada está representado como un subelemento dentro del elemento Add. El orden de los elementos x e y deben coincidir con el orden en el cual los parámetros son especificados en **la firma del método**. En otras palabras, colocar a y antes que x podría ser inválido. Además, los nombres y los tipos de los elementos Add, x e y deben ser los mismos que en el método destino y sus parámetros […] el cuerpo del mensaje de petición debe ser en el formato esperado por la aplicación remota.

¿Qué SOAP Response debe emitirse como respuesta en este sistema RPC? Después de haber creado un mensaje de petición propiamente formado, se debe observar la respuesta generada por la aplicación remota:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddResult>
      <result>3</result>
    </AddResult>
  </soap:Body>
</soap:Envelope>
```

El mensaje de respuesta regresado por la aplicación remota contiene el resultado del método Add. El parámetro de retorno es una vez mas codificado en un formato parecido a una estructura dentro del body del mensaje SOAP. La convensión de nombres del subelemento dentro del body es el nombre del método con Result anexado a este. Sin embargo, esta convensión en los nombres no ha sido dictada como una especificación. El primero (y en este caso, único) parámetro contiene el parámetro de regreso de la llamada al método. Como en el elemento AddResult, el nombre del elemento que contiene el parámetro de regreso no está dictado por esta especificación[247].

[247] Ibid., p. 40

Por otra parte, podríamos pensar acerca de cómo codificar los mensajes de un servicio Web que pase los parámetros por referencia, y no por valor. Podríamos partir éste análisis revisando la función que imprime una sección de la serie de Fibonacci , codificada en el mismo lenguaje C#[248]:

Un número en la serie de Fibonacci está determinado por añadir los dos números que le preceden directamente a este. Por ejemplo, si n1=1 y n2=1, n3=1+1=2, y n4=1+2=3. Aquí se muestra la rutina en C# que se utiliza para obtener la salida de los números en la serie de Fibonacci:

```
public void FibonacciIncrement(ref int n1, ref int n2)
{
    int temp = n2;

    // Set n1 and n2 to the next two Fibonacci numbers.
    n1 += n2;
    n2 = temp + n1;
}

// The following code prints the following output:
// 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
int x = 1;
int y = 1;

for(int i = 1, i < 11, i += 2)
{
    Console.Write("{0}, {1}", x, y);
    FibonacciIncrement(x, y);
}
```

En este caso, si a la función FibonacciIncrement se le dan los parámetros 1 1, regresará 1 2, si se le dan los parámetros 2 3 5 regresará 3 5 8, etc.

Para éste caso también es sencillo diseñar el formato de los mensajes del Web service. Para pasar los parámetros por referencia los mensajes se generarán de la siguiente manera[249]:

El método Web FibonacciIncrement acepta los últimos dos números y entonces regresa los siguientes dos números en la serie. Aquí se muestran los mensajes request y response para la primer llamada de FibonacciIncrement.

```
<!-- Request Message -->
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <FibonacciIncrement>
      <n1>1</n1>
      <n2>1</n2>
    </FibonacciIncrement>
  </soap:Body>
</soap:Envelope>


<!-- Request Message -->
<?xml version="1.0" encoding="utf-8"?>
```

[248] Ibid., pp. 49, 50
[249] Ibid.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <FibonacciIncrementResponse>
      <n1>2</n1>
      <n2>3</n2>
    </FibonacciIncrementResponse>
  </soap:Body>
</soap:Envelope>
```

Para finalizar esta sección, se invita a pensar en el sistema de correos tradicional. Para que un mensaje sea enviado de una persona a otra, no solo se requiere que esté bien escrito y propiamente ensobretado, identificado y sellado. También se requiere un medio de transporte, un agente de comunicaciones que lo mueva físicamente de una localidad a otra. En este sentido, los documentos propiamente codificados en SOAP pueden transmitirse utilizando diversos medios, pero el protocolo de transporte más popular utilizado para enviar los mensajes SOAP es HTTP. Sin embargo, los mensajes SOAP pueden también ser enviados a través de SMTP, vía fax, a una nave en el mar por radio de onda corta, o por cualquier otra vía imaginable[250]. De ello hablará el siguiente capítulo.

3.2.3 WSDL: Uso de XML para descripción de servicios

¿Cómo se definen los WSDL? Web Service Description Lenguaje (WSDL) es un dialecto basado en XML situado sobre los XML Schemas que definen un Web service. Un documento WSDL provee la información necesaria para el cliente a fin de interactuar con el Web service. WSDL es extensible y puede ser utilizado para describir prácticamente cualquier servicio en la red, incluyendo SOAP sobre HTTP y aún protocolos que no están basados en XML, como DCOM sobre UDP[251].

¿Porqué surgió esta tecnología? Debido a que se debe proveer un contrato WSDL para que los desarrolladores de clientes de Web services puedan utilizar los recursos distribuidos. Así, en un esfuerzo por describir como un cliente interactúa con un Web Service, usualmente se elabora un Schema para los mensajes que serán intercambiados entre el cliente y el servidor. El schema contiene una definición de tipo de dato compleja para los mensajes de solicitud y respuesta, para métodos distribuidos como Add o Substract. La meta final no es que los desarrolladores tengan solamente las puras definiciones de Schema, ni que traten de descifrar cómo interactuar con el Web service. En lugar de eso, siempre se piensa en describir el servicio de forma que hasta una sencilla herramienta pueda descifrarlo y crear un proxy en lugar del cliente[252].

Algunas de estas últimas herramientas son tan sencillas, que a veces no soportan tipos de datos complejos como arrays y structures. En estas herramientas –como el MSSoap Toolkit 3.0- regularmente se proveen mecanismos alternos, más directos, para construir el proxy soap del cliente. De cualquier manera se debe generar el WSDL para los demás métodos.

¿Qué más se requiere saber, aparte de los tipos de datos que acepta el Web service? Los desarrolladores preguntarán acerca de otros tipos de patrones de intercambio. En este sentido, además de utilizar la información provista por el Schema, WSDL se encarga de describir la forma en que los mensajes SOAP individuales pueden ser combinados para soportar una amplia variedad de patrones de intercambio de mensajes. Los patrones de intercambio de mensajes para el Web service son muy directos, pero una asociación formal entre los mensajes y sus mensajes de respuesta asociados podrían remover cualquier posible ambigüedad[253].

Una poderosa característica del lenguaje de descriptores WSDL es que dicho no está limitado para describir servicios de software, cuyos formatos de serialización están basados en XML. Esto significa que se puede usar WSDL para describir servicios que usan otros formatos, incluyendo los binarios. Por ejemplo, se puede usar WSDL para describir un servicio expuesto vía DCOM. En este caso, todavía se puede utilizar WSDL para describir los datos que están siendo utilizados a través del cable. La especificación WSDL provee las siguientes recomendaciones para efectuar esto:

---

[250] Ibid., p. 54
[251] Ibid., p. 98
[252] Ibid., p. 97
[253] Ibid.

- Describir los datos usando elementos, no atributos. Por ejemplo, cada parámetro debe estar codificado dentro de su propio elemento, de forma muy semejante en la codificación SOAP.
- Describir solamente los datos que están relacionados al mensaje y no son particulares a la codificación física. Por ejemplo, los parámetros pasados a un objeto COM remoto deben ser descritos en el schema. Sin embargo, el DCOM Object Identifier (OID) son datos específicos del protocolo de red física, que identifican el objeto y no deben ser descritos en el schema.
- Los tipos de arreglos deben ser derivados del tipo complejo Array definido en el SOAP Encoding schema. Por convención, el nombre del tipo debe ser el tipo de objetos dentro del arreglo, usando el prefijo ArrayOf.

Los parámetros que pueden contener datos de algún tipo deben ser definidos por un elemento de tipo xsd:anyType[254].

Además de esta ventaja, es obligatorio mencionar otra necesidad que no satisface el XML Schema por sí solo y que sí satisface el WSDL. Esto es una descripción formal de los patrones de mensaje, quien es aún más importante para Web services más complejos. Por ejemplo, algunos Web services podrían aceptar una petición pero no enviar una respuesta correspondiente de regreso al cliente. Otros podrían solamente enviar mensajes al cliente.

El Schema tampoco contiene información acerca de cómo utilizar el Web service. Debido a que SOAP es independiente de protocolo de transporte, los mensajes pueden ser intercambiados entre el cliente y el servidor en determinado número de formas. ¿Cómo se puede saber si se debe enviar el mensaje sobre http, SMTP o algún otro protocolo de transporte? Además, ¿Cómo puede saberse la dirección a la cual el mensaje debe ser enviado?[255]

Hasta este punto se ha hablado de muchas características generales de los WSDL. ¿Cuál es la anatomía de un documento WSDL? Los documentos WSDL pueden ser intimidantes a primera vista. Pero la sintaxis del documento WSDL no es más compleja que un documento XML Schema. Un documento WSDL está compuesto en series de asociaciones estratificadas sobre un documento XML Schema que describe un Web Service. Estas asociaciones aumentan la complejidad percibida de un documento WSDL. Pero una vez que se observan dichas asociaciones detenidamente, los documentos WSDL son muy directos[256].

¿Cómo es la estructura de un documento WSDL? La raíz de un documento WSDL es el elemento **definitions**. Dentro de este elemento existen cinco tipos de elementos hijos:

- **types.** Contienen las definiciones de schema de los mensajes que podrán ser enviados y recibidos por el servicio. La forma más común de representar el schema es usando un XML schema.
- **message**. Sirve como una referencia cruzada que asocia el mensaje con su definición dentro del schema.
- **portType** Define un conjunto de interfases que el Web service puede exponer. Una interfaz es asociada con uno o más mensajes.
- **binding.** Asocia la definición portType con un protocolo particular.
- **service** Define la colección de puntos finales relacionados (puertos) expuestos por el Web service[257].

¿Qué relación guardan los elementos Service, Binding, PortType, Message y Types? El siguiente diagrama ilustra cómo estos cinco elementos están estratificados sobre la definición de schema para describir el Web service:

[254] Ibid., p. 102
[255] Ibid., p. 98
[256] Ibid.
[257] Ibid.

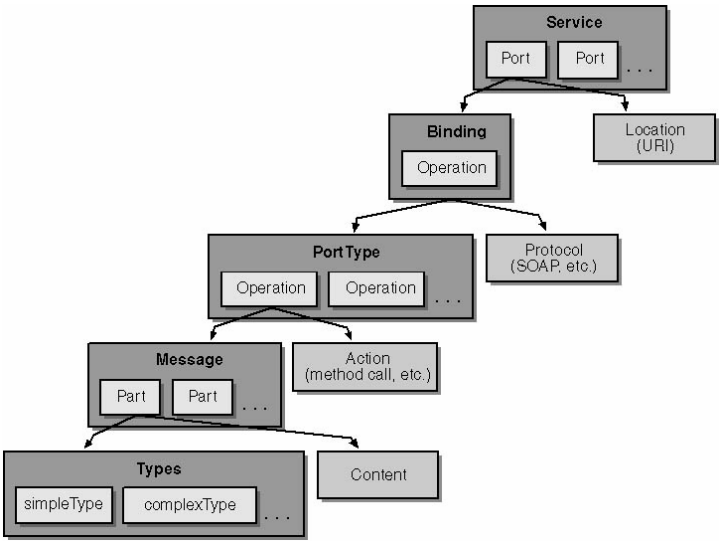


Ilustración 36 Estructura jerárquica de un documento WSDL

Como se puede ver, un documento WSDL está compuesto de series de asociaciones. Por ejemplo, las partes del mensaje son utilizadas para asociar una definición de tipos de datos con la porción del contenido del mensaje[258].

A pesar de que el diagrama es muy revelador, es insuficiente. Por ello, es necesario exponer la funcionalidad de cada uno de estos elementos, para obtener una clara idea del documento WSDL.

En primer lugar, ¿Cuál es la función del elemento WSDL Definitions? El elemento raíz en un documento WSDL, el elemento *definitions*, sirve de la misma forma que el elemento *schema* en un documento XML schema. Contiene elementos hijos que definen un servicio en particular.

De forma muy semejante a un documento XML Schema, un documento WSDL puede definir su propio namespace añadiendo el atributo *targetNamespace* al elemento *definitions*. La única restricción es que el valor del atributo *targetNamespace* no puede contener un URI relativo.

El namespace de WSDL permite calificar completamente las referencias a entidades definidas dentro del documento WSDL. Por ejemplo, una definición de mensaje está referida por una definición *portType*.[…] esta característica facilita la herencia en las interfases[259].

El elemento *definitions* define los límites de un alcance de nombres en particular. Elementos declarados dentro del documento WSDL son utilizados para definir entidades tales como puertos y mensajes. Estas entidades son asignadas a un nombre usando el atributo *name*. Todos los atributos de nombre dentro del alcance de nombres deben ser únicos. Por ejemplo, si un documento WSDL contiene un puerto llamado *Foo*, no puede contener otro puerto o mensaje llamado *Foo*[260].

En segundo lugar ¿Para qué sirve el elemento WSDL Types? Dicho es utilizado para determinar los tipos de datos referenciados por el WSDL. El elemento types contiene información del schema referenciado dentro del documento WSDL. **El sistema de tipos por defecto soportado por WSDL es XML Schema**. Si XML Schema es utilizado para definir los tipos contenidos dentro del elemento types, el elemento *schema* aparecerá como un elemento hijo inmediato.

[258] Ibid., p. 99
[259] Ibid.
[260] Ibid., p. 100

Se puede usar otros sistemas de tipos por extensión. Si se está usando otro sistema de tipos de datos, un elemento de extensibilidad podrá aparecer bajo el elemento *types.* El nombre del elemento debe identificar el sistema de tipos utilizado. XML Schema es el sistema de tipos de datos dominante utilizado en documentos WSDL, incluyendo aquellos para los Web services desarrollados en la plataforma .NET[261].

Los XML Schemas se utilizan no solo para describir tipos de datos, sino también también para validar documentos, en conjunción con herramientas como Xerces XSD, o MSV (Multi Schema Validator), el cual se utilizará más adelante en la tesis.

En tercer lugar, ¿Cuáles son los propósitos del elemento WSDL Message? El elemento Message provee una abstracción común para mensajes pasados entre el cliente y el servidor. Debido a que se pueden utilizar múltiples formatos de definición de Schema dentro de un documento WSDL, es necesario tener una forma común de identificar los mensajes. El elemento Message provee este nivel común de abstracción que será referenciado en otras partes del documento WSDL[262].

¿Cómo está compuesto el elemento WSDL Message? Múltiples elementos *message* pueden y usualmente aparecen en un documento WSDL, uno para cada mensaje que está siendo comunicado entre el cliente y el servidor. Cada mensaje contiene uno o más elementos *part* que describen piezas de contenido dentro del mensaje. Un ejemplo de una parte es el cuerpo de un mensaje SOAP o un parámetro contenido dentro del query string, un parámetro codificado en el cuerpo de un mensaje SOAP, o el cuerpo entero de un mensaje SOAP.

Cada elemento *part* contiene atributos que asocia definiciones type y element, encontradas en el elemento types. Debido a que partes son definiciones abstractas del contenido, la información de binding debe ser examinada a fin de determinar el significado de las partes.

Dos atributos que pueden aparecer dentro del elemento part son los atributos *element* y *type*. El atributo *element* se refiere a una definición de elemento en un schema. El atributo *type* se refiere a una definición de tipos de datos en el schema[263].

Para concluir el análisis de este nodo, se debe destacar que las partes de un mensaje pueden ser tomadas de diversas fuentes. Debido a que cada parte puede servir como una definición abstracta de una pieza de datos, un mensaje puede estar compuesto de múltiples piezas de datos de múltiples fuentes. Aunque no es recomendado, se podría describir un mensaje en el cual algunos de los parámetros fueran codificados dentro del cuerpo de SOAP y algunos de los parámetros fueran codificados dentro del query string[264].

En cuarto lugar, ¿Qué es el elemento WSDL PortType? El elemento portType contiene un conjunto de operaciones abstractas representando los tipos de correspondencias que pueden ocurrir entre el cliente y el servidor. Para Web services de estilo RPC, un portType puede ser considerado como una definición de interfaz en la cual cada método puede ser definido como una operación[265].

¿Qué tipos operaciones abstractas son las descritas con PortType? Un portType está compuesto de un conjunto de elementos *operation* que define una acción en particular. Los elementos *operation* están compuestos de los mensajes definidos dentro del documento WSDL. WSDL define cuatro tipos de operaciones, conocidos como *operation types*:

- **Request-response** La comunicación estilo RPC en la cual el cliente hace una petición y el servidor atiende la respuesta correspondiente.
- **One-way** La comunicación estilo documento en la cual el cliente envía un mensaje pero no recibe una respuesta del servidor indicando el resultado de un mensaje procesado.

[261] Ibid.
[262] Ibid., pp. 102, 103
[263] Ibid., p. 103
[264] Ibid., p. 104
[265] Ibid.

- **Solicit-response** Lo opuesto a la operación request-response. El servidor envía la solicitud, y el cliente envía de regreso la respuesta.
- **Notification** Lo opuesto a la operación one-way. El servidor envía las comunicaciones estilo documento al cliente[266].

¿Cómo, exactamente, se encuentra estructurado el elemento PortType? Una operación está compuesta de un subconjunto de elementos input, output y fault. El tipo de elementos y el orden de los elementos dentro de la operación determinan el tipo de operación. Por ejemplo, one-way define un mensaje de entrada, y request-response define un mensaje de entrada y un mensaje de salida. Los tipos de operación solicit-response y notification son lo opuesto de request-response y one-way, respectivamente. La operación solicit-response lista el mensaje de salida y entonces el mensaje de entrada, y la operación notificación contiene un mensaje de salida en lugar de un mensaje de entrada[267].

¿En qué orden se encuentran los mensajes input, output y fault? La tabla lista el tipo y el ordenamiento de los mensajes para cada tipo de operación:

Operation Type	Input	output	fault
Request-response	1	2	3*
One-way	1		
Solicit-response	2	1	3*
Notification		1	

* El mensaje fault es opcional. Cualquier número de mensajes fault pueden aparecer en una operación

Tabla 20 Orden de los mensajes para los tipos de operación

¿Cómo se declaran los mensajes de falla en el elemento WSDL PortType? Las operaciones que invocan comunicación de dos vías pueden especificar opcionalmente uno o más mensajes de falla. Así como en las definiciones de método de Java, los mensajes de falla permiten declarar el tipo de excepción que puede ser lanzada por la aplicación en el servidor. Sin embargo, la lista de posibles fallas no debe incluir errores que están especificados por el protocolo de transporte subyacente. Por ejemplo, no sería deseable representar el error HTTP 500 dentro del documento WSDL

En las operaciones existe una convención para los nombres. Los nombres de los elementos input, output y fault tienen un valor por default si uno no es especificado. Para los tipos de operación one-way y notification, el nombre por default es el nombre del elemento operation, el nombre de los elementos input y output por default con el nombre de la operación con *Request* o *Response* anexa al final. Para solicit-response, el nombre del elemento output es por default el nombre de la operación con *Solicit* o *Response* anexo al final.

Debido a que múltiples elementos *fault* pueden ser definidos dentro de la operación, no existe un nombre por default para el elemento *fault*. Sin embargo, cada elemento fault debe ser llamado de manera única dentro de su elemento *operation* padre[268].

En quinto lugar, ¿qué propósito tiene el elemento WSDL Binding? El elemento binding contiene definiciones de asignación para asociar un protocolo como SOAP a un bindingType. Las definiciones *binding* especifican el formato del mensaje y detalles de protocolo. Por ejemplo, la información de asignación específica si se puede utilizar una instancia de un portType en una forma parecida a RPC.

Las definiciones *binding* también indican el número de comunicaciones en red requeridas para realizar una acción particular. Por ejemplo, una llamada SOAP RPC sobre http podría envolver un intercambio de comunicaciones http, pero esa misma llamada sobre SMTP podría envolver dos intercambios de comunicaciones SMTP discretos[269].

[266] Ibid.
[267] Ibid., p. 105
[268] Ibid.
[269] Ibid., pp. 106, 107

Los bindings son ejecutados a través del uso de elementos extensión. Cada protocolo tiene su propio conjunto de elementos extensión para especificar los detalles del protocolo y el formateo de los mensajes. Para un protocolo en particular, los elementos extensión son usualmente utilizados para decorar las acciones individuales dentro de una operación y la operación por si misma con la información de asignación de protocolo. Algunas veces, los elementos extensión son utilizados en el nivel portType por si mismo[270].

En sexto y último lugar, ¿Qué información se encuentra en la Definición WSDL Service? Un servicio es un grupo de puertos relacionados y es definido por el elemento service. Un puerto es un punto final particular para el Web service que es referenciado por una dirección simple. Los puertos definidos dentro de un servicio particular son ortogonales. Por ejemplo, la salida de un puerto nopuede servir como la entrada de otro[271].

¿Qué estructura tiene dicho elemento? El elemento service es utilizado para agrupar un conjunto de puertos relacionados. Un puerto contiene un elemento extensión que provee la dirección en la cual está localizado. Si se necesita especificar más de una dirección, se debe crear un puerto para cada dirección. Si se definen múltiples puertos del mismo portType (y posiblemente diferentes direcciones) dentro del mismo Web service, estos deben ser considerados alternativos.

Estos, además, deben proveer el mismo comportamiento, pero sobre diferentes protocolos de transporte. El cliente puede iterar a través de los puertos para buscar un binding compatible con un portType apropiado, y un protocolo[272].

Estos seis elementos, descritos anteriormente a grandes rasgos, no son los únicos bloques de construcción de un Web service. Adicionalmente a ellos, existen elementos de extensibilidad en WSDL, y proveen información adicional para el Binding protocol de transporte. Los elementos de extensibilidad son utilizados para representar tecnologías particulares. Por ejemplo, se pueden usar elementos de extensibilidad para especificar el lenguaje de schema utilizado dentro del elemento types.

El schema para un conjunto particular de elementos de extensibilidad debe ser definida dentro de un namespace diferente al del WSDL. La definición de los elementos por sí mismos pueden contener el atributo *wsdl:requires* que especifica un valor booleano. Si el atributo *required* es establecido en *trae* dentro de una definición de elemento, un binding que hace referencia al conjunto particular de elementos de extensibilidad deben incluir ese elemento.

Más comúnmente, elementos de extensibilidad son utilizados para especificar información de binding. La especificación WSDL define conjuntos de elementos de extensión para enlazar con SOAP, HTTP GET, HTTP POST, y MIME. No obstante, la especificación define los enlaces para solamente dos de los cuatro tipos de operación, one-way y request-response[273].

El único elemento de extensión http especificado dentro del elemento *service* es *http:address.* Así como su contraparte SOAP, está contenido dentro de la definición de puerto y es utilizada para especificar el URO en donde el servicio Web puede ser alcanzado.

Estos bindings del elemento service son utilizados para notificarle a los desarrolladores en donde se encuéntra el daemon, el servlet o el cgi que escucha las peticiones HTTP-POST de SOAP[274].

Para resumir, se puede concluir que los WSDL permiten desarrollar con un buen grado de modularidad los Web services. Tal como los documentos XML Schema, los documentos WSDL pueden importar otros documentos. Se puede sin embargo alcanzar el mismo nivel de modularidad que se tiene en los documentos XML Schema. Debido a que un documento WSDL puede llegar a ser muy grande muy rápidamente, romperlo en un número de documentos pequeños puede ayudar a hacer el documento fácil de entender y posiblemente fácil de mantener.

[270] Ibid., p. 107
[271] Ibid., p. 108
[272] Ibid., p. 109
[273] Ibid.
[274] Ibid., p. 119

Una forma común de dividir una definición de un servicio sencillo en múltiples documentos WSDL es colocar la información del protocol binding en un documento por separado. Esto permite escribir las definiciones de interfaz una sola vez y entonces importarlas dentro un documento WSDL que defina los protocolos específicos soportados por una instancia en particular del Web service[275].

¿Qué formas existen de publicar un Web service, descrito propiamente a través de un WSDL? La siguiente sección comentará brevemente las maneras de publicar un Web Service.

[275] Ibid., p. 120

3.3 Métodos de localización de los Web Services

Las compañías necesitan una forma para publicar los Web services que soportan, y para los clientes a fin de que descubran dichos servicios. Para tal efecto existen dos formas de mecanismos para el descubrimiento de los Web services: Universal Description, Discovery, and Integration (UDDI) y DISCO. UDDI es un servicio de directorio central y jerárquico; DISCO promueve un modelo más libre para localizar los Web services[276].

¿Qué es UDDI? UDDI provee un servicio de directorio central para publicar información técnica acerca de los Web services. UDDI es el resultado de una iniciativa de la industria respaldada por un número significativo de compañías tecnológicas, incluyendo Microsoft, IBM, y Ariba.

UDDI es todavía otro ejemplo de un nivel sin precedentes de la cooperación entre industrias alrededor de la adopción de los Web services. Muchas compañías creyeron que un servicio de directorio para publicación de Web services podría ser crucial para que los Web services ganaran aceptación masiva, y estos sintieron que el tiempo necesario para desarrollar la especificación UDDI a través de los cuerpos estándar fue inaceptable.

Como resultado de la cooperación entre compañias, se generó la especificación UDDI y la infraestructura requerida para desarrollarla y soportarla. Una vez que UDDI haya alcanzado un "razonable nivel de madurez", los miembros de proyecto trasladarán dichos lineamientos al cuerpo de estándares formales[277].

¿Qué elementos básicos tiene UDDI? La infraestructura que soporta UDDI está compuesta de un conjunto de registries y registrars. Un registry contiene una copia completa del directorio UDDI; un registrar provee los servicios de registro de UDDI en representación del usuario del Web service.

Un registrar puede ser un ISP, un servidor de business-to-business (B2B) para el mercado de negocios, una compañía individual, o el servidor del registry por si mismo. Por ejemplo, Microsoft ofrece un registry y también provee una interfaz de usuario HTML para crear y mantener los registros dentro del directorio[278].

¿Cómo están relacionados estos elementos básicos? Y lo más importante, ¿Cuál es la meta del modelo UDDI? UDDI es un modelo que se basa en crear duplicados de la información en diversos servidores registry. En sí, los registries están basados en un modelo de duplicado single-master. Esto es, un negocio debe escoger un registry en el cual pueda mantener su información. Todas las actualizaciones hechas al directorio deberán ser duplicadas para los demás registros. Entonces la información actualizada puede ser consultada por algún registry.

[276] Ibid., p. 256
[277] Ibid.
[278] Ibid.

El siguiente diagrama muestra a un usuario actualizando el directorio de negocios UDDI a través de un registrar y entonces otro usuario puede utilizar la información actualizada[279].

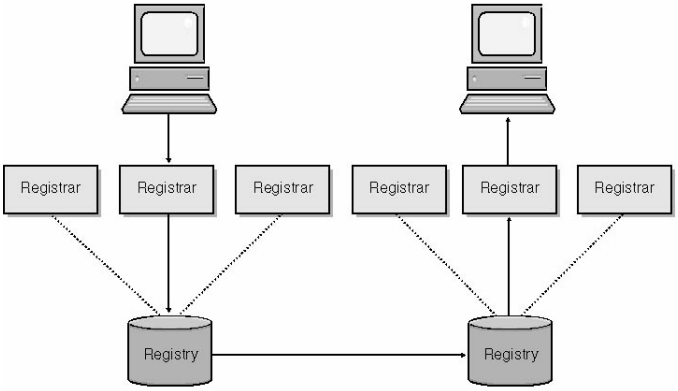


Ilustración 37 Un usuario actualiza el directorio de negocios UDDI. Otro usuario utiliza la información actualizada

Visto más de cerca, ¿qué es un UDDI registry? Un UDDI registry por si mismo es un Web service. Expone una API basada en SOAP para utilizar y manipular información dentro del directorio. En lugar de mantener datos a través del registrar, un desarrollador puede programar directamente contra la API[280].

¿De qué forma se generan las peticiones al UDDI registry? La versión 1 de la UDDI API expone cerca de 30 métodos para interactuar con un registry, el cual se comporta de forma síncrona. El siguiente ejemplo muestra como un mensaje de UDDI request está estructurado[281]:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_business generic="1.0" xmlns="urn:uddi-org:api">
      <name>MyTestBusiness</name>
    </find_business >
  </Body>
</Envelope>
```

Al ser la especificación UDDI un lineamiento de un Web service en particular, no nos sorprende entonces que también utilice los XML SOAP Messages, como los Web services "normales" o "estándar".

¿Qué requisitos especiales debe reunir el mensaje de UDDI? Algunos de esos elementos requeridos son los siguientes[282]:

- La especificación UDDI requiere que el mensaje SOAP sea codificado en UTF-8
- Los elementos dentro del body del documento UDDI deben estar dentro del alcance del namespace del UDDI API. El namespace del UDDI AP es identificado por el URI *urn:uddi-org:api.*
- La petición debe contener un encabezado SOAPAction http cuyo valor es una cadena vacía. Algún otro valor será considerado como un error.
- La versión del API destino debe fijarse dentro del cuerpo del mensaje usando el atributo *generis.*

[279] Ibid., p. 257
[280] Ibid.
[281] Ibid.
[282] Ibid., p. 258

Ahora bien, todo Web service tiene un conjunto finito de métodos. ¿Cuáles son estos en el API de UDDI? Los métodos del UDDI API pueden dividirse en dos categorías: los métodos de consulta y los métodos de publicación[283]. Los métodos de consulta permiten buscar y navegar en el directorio, y los métodos de publicación permiten modificar el contenido del directorio. Los mensajes de los métodos de consulta tienen un elemento root en el SOAP body, con el prefijo find_ o_get_. Con un par de excepciones, los mensajes de los métodos de publicación tienen un elemento raiz en el cuerpo del mensaje SOAP con el prefijo save_ o delete_.

En primer lugar, ¿Qué métodos de consulta existen en el API de UDDI? A continuación se da una tabla con los métodos básicos de este tipo[284].

Message	**Description**
find_binding	Searches for bindingTemplate entities that match a specified set of criteria
find_business	Searches for businessEntity entities that match a specified set of criteria
find_service	Searches for businessService entities that match a specified set of criteria
find_tModel	Searches for tModel entities that match a specified set of criteria
get_bindingDetail	Obtains one or more specific bindingTemplate entities
get_businessDetail	Obtains one or more specific businessEntity entities
get_businessDetailExt	Obtains one or more specific businessEntityExt entities from a registry that may support additional attributes
get_serviceDetail	Obtains one or more specific businessService entities
get_tModelDetail	Obtains one or more specific tModel entibies

Tabla 21 Métodos de consulta existentes en el API de UDDI

Los métodos find_ son para búsquedas generales, y los métodos get_ son para obtener información detallada acerca de un registro en particular.

En segundo lugar, ¿Qué métodos de publicación están en el API del UDDI? Se pueden citar los siguientes[285]:

Message	**Description**
delete_binding	Deletes one or more specified bindingTemplate entities
delete_business	Deletes one or more specified businessEntity entities
delete_service	Deletes one or more specified businessService entities
delete_tModel	Deletes one or more specified tModel entities
discard_authToken	Invalidates the previously obtained authentication token
get_authToken	Obtains an authentication token for the user
get_registeredInfo	Obtains a list of businessInfo and tModelInfo entities for a particular user
save_binding	Saves one or more bindingTemplate entities
save_business	Saves one or more businessEntity entities
save_service	Saves one or more businessService entities
save_tModel	Saves one or more tModel entities

Tabla 22 Métodos de publicación en el API del UDDI

Los métodos de publicación son responsables de manipular datos dentro del directorio. La creación y actualización es manejada por los mensajes save_. Cuando un registro es creado usando el método save_, el registry genera un identificador único. El identificador único es entonces pasado a llamadas subsecuentes al método save_ para actualizar el registro. Cada mensaje save_ tiene un mensaje delete_ correspondiente para eliminación de información.

¿Existen mecanismos de seguridad en UDDI? Las modificaciones realizadas usando los métodos de publicación deben ser efectuadas de forma segura. La especificación UDDI establece que todos los mensajes de publicación deben ser

[283] Ibid., p. 258
[284] Ibid., p. 259
[285] Ibid., pp. 259, 260

intercambiados entre el cliente y el servidor sobre HTTPS. Esto previene que el contenido del mensaje sea modificado durante el tránsito.

Un registro UDDI permite modificar y borrar solamente los propios registros de usuario, por lo que se debe incluir un token de identificación con cada petición para probar la identidad. Este token es pasado como parte de la firma del método de los métodos save_ y delete_.

Se obtiene un token de identificación transfiriendo las credenciales al método get_authToken. Los detalles de implementación sobre cómo un registry crea el token de identificación se dejan al registry. El registry de Microsoft utiliza Passport para identificar sus usuarios. Se transfiere un conjunto válido de credenciales de Passport para un usuario registrado, y el registry regresa un token válido de Pasaporte[286].

¿Qué tipos de datos maneja UDDI? Los cuatro tipos de datos primarios son tModel, businessEntity, businessService, y bindingTemplate. UDDI provee una API para publicar y localizar instancias de estos tipos de datos[287].

- **tModel** sirve con un propósito doble. Primero, se puede utilizar para referenciar una especificación técnica tal como un documento WSDL, un protocolo de transporte, o especificación de flijo. Segundo, un tModel puede referenciar una taxonomia particular utilizada para clasificar la información publicada dentro del directorio.
- Un **businessEntity** es comúnmente una compañía o una división dentro de una compañía. Este contiene información tal como la dirección de la compañía y la información del contacto. Esto también contiene información que clasifica la compañía, tal como una industria particular.
- Un **businessService** es una colección de servicios relacionados que una compañía expone. Un businessService está compuesto de una colección de objetos bindingTemplate.
- Un **bindingTemplate** describe un servicio particular, incluyendo el punto final del servicio y las especificaciones técnicas que el servicio soporta.

Cada firma de software tiene sus propias herramientas de software para implementar e interactuar con los mecanismos de descubrimiento de los Web services. En el caso de Microsoft, para implementar e interactuar con UDDI y DISCO dispone de lo siguiente[288]:

Microsoft provee el UDDI SDK para facilitar el desarrollo de aplicaciones equipadas con UDDI. El SDK contiene un modelo de objetos .NET para simplificar el contacto con un registro UDDI. Este también contiene UDDI Developer Edition, un registry standalone que es instalado localmente y que soporta completamente la UDDI API.

DISCO es un mecanismo ligero para descubrir Web services. Este es utilizado primeramente para desarrollo del Visual Studio .NET IDE. Visual Studio .NET automáticamente crea los archivos DISCO necesarios. El archivo DISCO para un servidor en particular es creado en tiempo de instalación; el archivo DISCO para un Web service particular es creado a través del proyecto Visual Studio .NET en el cual es contenido.

[286] Ibid., p. 260
[287] Ibid., p. 288
[288] Ibid.

3.4 XML Web services: Envío síncrono y asíncrono de XML usando el protocolo de transporte HTTP

Con la llegada de las mini computadoras y las computadoras personales, la descentralización del procesamiento y el almacenamiento de los datos llegó a ser algo deseable. Sin embargo, aún cuando el procesamiento y almacenamiento de datos ya no está físicamente centralizado, la aplicación de forma lógica todavía sigue siendo una aplicación, al diseñarla de manera distribuida. Una aplicación distribuida es aquella cuyos requerimientos de procesamiento pueden ser satisfechos por múltiples computadoras físicas, y cuyos datos pueden ser almacenados en muchas localidades físicas, pero dichas funciones lógicas no están determinadas por la topología física que es utilizada para programar la aplicación[289].

¿Por qué se necesitan las aplicaciones distribuidas? Las fuerzas gobernantes detrás del procesamiento descentralizado y almacenamiento de datos incluye:

- **Costo de los servidores Mainframe**. Una de las fuerzas primarias que impulsó las aplicaciones distribuidas fue el costo de los mainframes. No solo fue el costo de la inversión inicial mas allá del alcance de la mayoría de las compañias, pero teniendo un simple punto de falla fue un riesgo que la mayoría de las compañias del cual no podían darse el lujo.
- **Pertenencia de los datos**. Un factor importante detrás de la descentralización fue las políticas de pertenencia de los datos. Departamentos, divisiones, localidades geográficas o lugares que poseen los datos no desean delegar la responsabilidad de administrar sus datos a otra localidad central.
- **Seguridad**. Otro factor importante fue la seguridad. Para una organización, típicamente la mayoría de sus datos necesitan ser fácilmente accesibles. Sin embargo, datos corporativos sensibles todavía deben ser asegurados. Tratar con estos dos requerimientos de seguridad fue mas fácil cuando los datos pudieron ser segmentados físicamente[290].

¿De que forma actúan los componentes distribuidos? Se dice que cada uno de estos bloques de construcción es un proveedor de servicios. Esto es, con la aparición del patrón de diseño para las aplicaciones distribuidas vino la comprensión de que la industria todavía no había alcanzado su meta en el reuso. En lugar de ver las aplicaciones distribuidas como lógicamente monolíticas, llegó a ser muy útil ver los componentes distribuidos de aplicaciones como proveedores de servicios para una aplicación lógica. El concepto de funcionalidad distribuida retuvo la premisa del reuso. Los desarrolladores pudieron usar cada uno de los conjuntos distribuidos de funcionalidad como bloques de construcción para aplicaciones mucho más grandes. Los problemas significativos que conllevaba el reuso de componentes distribuidos fueron abatidos desde la adopción de las nuevas tecnologías que trajo Internet.

Aunque la Internet ha existido por más de veinte años, fue solamente a mediados de 1990 que la posibilidad de la Internet proveyó la infraestructura significativa para construir aplicaciones distribuidas. Los protocolos simples basados en texto fueron desarrollados como los medios primarios para comunicar solicitudes de servicio y enviar datos sobre Internet. La adopción popular de tales protocolos hizo de Internet una plataforma viable para aplicaciones distribuidas. En lugar de confiar en tecnologías siempre competitivas y la mayoría de las veces propietarias, los estándares de Web formaron los fundamentos de aplicaciones distribuidas para la Web[291].

¿Qué consideraciones de diseño se tienen al momento de programar aplicaciones distribuidas basándose en los estándares de Web? Existen muchos problemas comunes que es necesario considerar cuando se diseña una aplicación distribuida. Estos problemas no son únicos a algún diseño de aplicaciones distribuidas en particular. Dichos problemas son:

- **Tipos de datos no compatibles a través de diferentes sistemas**. Diferentes sistemas operativos soportan diferentes tipos de datos. Algunas veces, no existe compatibilidad al 100% en los tipos de datos a través de

[289] MICROSOFT CORPORATION. "XML Web Service Architectures" En: Developing XML Web Services Using Microsoft© ASP.NET, p. 2
[290] MICROSOFT CORPORATION. "The Need for XML Web Services" En: Developing XML Web Services Using Microsoft© ASP.NET, p. 3
[291] Ibid.

diferentes sistemas operativos. Sin embargo, se debe considerar como manejar los tipos de datos que no son compatibles a través de diferentes sistemas.

- **Fallas del servidor o pérdida en las respuestas del servidor**. Debido a que los componentes de aplicaciones distribuidas son comúnmente remotos, existen más puntos de falla aislados. La falla en algún punto aislado puede causar que la aplicación distribuida completa falle. No obstante, se debe considerar cómo manejar las fallas en el servidor y la pérdida de comunicación con el mismo.
- **Fallas en el cliente**. Si un servidor está almacenando el estado en representación del cliente, y el cliente falla, entonces se debe considerar cómo será notificado el servidor. Se debe considerar también si es necesario reclamar los recursos en el servidor que fueron utilizados por el cliente.
- **Reintento de llamada**. Si un método remoto es llamado y no hay respuesta desde el servidor, no será aceptable reintentar llamar a ese método. Por ejemplo, si un método es llamado para comprar una gran cantidad de artículos para inventario, y si el servidor recibió la solicitud para colocar la orden, pero la respuesta se perdió, ¡no sería aceptable reenviar la orden de compra!
- **Seguridad**. En las aplicaciones distribuidas, existen más oportunidades para aplicar capas de seguridad. No solo se debe considerar la identificación del usuario y su rol de autorización, sino que se debe considerar como proteger el canal de comunicaciones entre el cliente y el servidor, y cómo defenderse contra ataques de espionaje de señal, ataques de negado de servicio, ataques de repetición, y así por el estilo.
- **Sincronizar relojes entre múltiples computadoras**. Muchas operaciones parten del registro de tipos de dato fecha. Por ejemplo, no es aceptable para el servidor el recibir una notificación de compra antes que la orden de compra sea recibida. Este tipo de problema puede tomar lugar si los relojes en el cliente y el servidor no están sincronizados. Sin embargo, se debe decidir cómo se asegurará la sincronización de los relojes en varias computadoras que se comunican en una aplicación distribuida[292].

Se ha hablado por separado de los sistemas distribuidos y de los Web services. ¿Qué relación existe entre ellos? Los XML Web services son los bloques de construcción fundamentales en la transición al cómputo distribuido sobre Internet. Los estándares abiertos y la atención sobre la comunicación y colaboración entre personas y aplicaciones ha creado un ambiente en el cual los XML Web services han llegado a ser la plataforma seleccionada para integrar aplicaciones. Las aplicaciones son construidas usando múltiples XML Web services desde diferentes fuentes que trabajan juntas sin considerar donde residen o cómo fueron implementadas[293].

Los modelos de objetos distribuidos tales como DCOM (Distributed Component Object Model), Java Remote Method Invocation (RMI) y CORBA (Common Object Request Broker Architecture), sufren de la limitante de estar apoyados en protocolos binarios. Algunos de los problemas inherentes al usar los protocolos binarios incluyen:

- **Firewalls**. El primer problema es que los protocolos binarios son 100% punto a punto. Como resultado, alguna comunicación con el punto final es dentro del dominio del firewall, por lo que se requiere administradores de firewall para abrir un rango de puertos para permitir las comunicaciones. Para la mayoría de las organizaciones, esto es un riesgo de seguridad inaceptable.
- **Interoperabilidad**. Otro problema es la interoperabilidad entre diferentes modelos de objetos. Cada modelo de objeto utiliza su propio protocolo propietario. Es posible proveer software para traducir los paquetes que pasan entre los diferentes modelos de objetos. Sin embargo, una pérdida de información siempre resulta debido a la traducción. El resultado es que la mayoría de las organizaciones usan el modelo de objetos simple para compilar todos sus sistemas dentro de la organización. Consecuentemente, el entorno de aplicaciones distribuidas está dividido en diferentes grupos que son identificados por el modelo de objetos que cada grupo ha adoptado. Si un socio de comercio potencial escoge un modelo de objetos competitivo, este eventualmente causará problemas significativos.
- **Formatos de datos**. Otro problema con los protocolos binarios es la codificación de datos que son transmitidos, usando esos protocolos. Cada protocolo codifica los datos de forma diferente, lo cual coloca una enorme sobrecarga en organizaciones cuando estas tienen que consumir los datos que son codificados en múltiples y

[292] Ibid., p. 5
[293] MICROSOFT CORPORATION. "Introduction to SOAP and XML Web Services" En: Building XML- Based Web Applications, p. 1

frecuentemente, incompatibles formas. También, la dificultad de traducir los datos de un formato a otro dirige la segregación de las organizaciones basadas en los formatos de datos que pueden manejar[294].

Los problemas con los modelos de objetos existentes para aplicaciones distribuidas (RMI, CORBA, DCOM, etc) han orillado a los desarrolladores a buscar nuevas alternativas. Con la rápida adopción de los estándares de Web, es natural el que las soluciones basadas en estándares de Web fueran consideradas. Esto dirige la evolución de los web services.
Los Web services:

A. Permiten que las aplicaciones se comuniquen usando HTTP sobre TCP/IP (World Wide Web o Internet) lo que los hace amigables con todo tipo de **firewall**.
B. Son unidades de aplicación basadas en XML que proveen datos y servicios a otras aplicaciones, sin considerar el sistema operativo o lenguaje de programación. Esto también agrega **interoperabilidad** al proveer un **sistema homogeneo de representación de datos**.

Tratando el punto A, debe iniciarse con el comentario de que, debido a los problemas de utilizar protocolos binarios, se necesitó un protocolo omnipresente, fácil de entender por los humanos y los programas, con datos fáciles de codificar y transformar. Este protocolo hizo que la aparición del World Wide Web proveyera la solución universal que todos pudieran utilizar fácilmente.

Las especificaciones Transmission Control Protocol (TCP) e Internet Protocol (IP) fueron desarrolladas originalmente para conectar diferentes redes de diferentes diseñadores en una red de redes. Últimamente, dicha red de redes ha llegado a tomar el nombre de Internet.

Entonces, a finales de 1990, Tim Berners-Lee, un científico de la computación en CERN inventó el World Wide Web, el que es conocido como Web. La Web es una red global interconectada de documentos de hipertexto. A partir de este esfuerzo, emergieron dos tecnologías revolucionarias: Hypertext Markup Language (HTML) e Hypertext Transfer Protocol (HTTP).

HTML es un lenguaje que define como añadir marcas (en forma de etiquetas) a los documentos de texto para proveerle información al Web Browser sobre como formatear el texto en el documento. Los documentos con etiquetas HTML son conocidos como documentos de hipertexto[295].

¿Qué ventajas ofrece HTTP? HTTP es el protocolo que es utilizado para solicitar y recibir documentos de hipertexto en la Web. Un punto muy importante a notar acerca de HTTP es que no está restringido para trabaja solamente con documentos HTML. Un ejemplo de este hecho son los XML Web Services y sus clientes pueden intercambiar documentos XML utilizando HTTP.

En la medida que se incrementó la popularidad del Web, http como protocolo ha sido adoptado casi universalmente. Usando HTTP se abate uno de los mayores obstáculos de la interoperabilidad de los modelos de objetos distribuidos, específicamente la falta de un protocolo omnipresente y confiable[296].

Además, los servidores Web tipicamente se comunican utilizando HTTP, el cual es un protocolo ampliamente aceptado por ser confiable. Un aspecto igualmente importante del Web Server es su papel como "gateway" en una organización. Los servidores Web necesitan no solamente servir contenido HTML, sino que a través de los mecanismos de extensibilidad HTTP, los servidores Web pueden también reenviar solicitudes al manejador de solicitudes apropiado.

El servidor Web no necesita darse cuenta por sí mismo sobre cómo el manejador interpreta la carga de datos de una solicitud HTTP.Esto es porque es la responsabilidad del manejador el procesar la solicitud recibida y generar una respuesta HTTP. El servidor Web envía la respuesta de regreso al cliente. Los servidores Web pueden reenviar las solicitudes del

[294] Microsoft Corporation. The Need for XML Web Services... pp. 11,12
[295] Ibid., p. 12
[296] Ibid.

cliente a algun tipo de servicio que una solicitud HTTP describe, y cuyo resultado puede ser a su vez codificado en una respuesta http. Todo esto puede efectuarse sin necesitar alguna reconfiguración, o perder una política en el Firewall.[297]

Para comentar el punto B, se plantea la pregunta ¿Qué ventaja ofrece XML, por su parte? XML es un formato universal de datos. Los desarrolladores se dieron cuenta rápidamente que aunque HTML permitía al autor de un documento definir estructura en la presentación, no proveía alguna manera de definir la estructura de los datos o las relaciones entre los datos en un documento. En 1996, ésta limitante dirigió el nacimento de un lenguaje para marcar texto y describir la estructura de los datos dentro del documento. Este lenguaje es conocido como Extensible Markup Language (XML). Algunas de las metas de los documentos XML son que deben ser:

1. Fácilmente utilizables en Internet
2. No ambiguos
3. Fáciles de crear
4. Fáciles de interpretar y procesar
5. Extensible, de plataforma independiente y con la posibilidad de soportar localización.

La rápida adopción de XML es evidencia de su utilización como formato universal de datos[298].

La Web aún tiene ciertos problemas que no se han resuelto. A pesar de los beneficios que la Web ha provisto, tales problemas abarcan las grandes áreas de seguridad y rendimiento:

- **Seguridad**. Debido a que Internet es una infraestructura publica, este también implica que cualquier comunicación es potencialmente vulnerable a la intercepción, modificación, spoofing (técnica que es utilizada para ganar acceso no autorizado en una computadora), etc.

- **Rendimiento**. La mayoría de los usuarios de Internet todavía tienen acceso dial-up a Internet. Esto introduce problemas de rendimiento significativos y restricciones severas en el tipo y complejidad en que la aplicación puede ser entregada sobre la Web. Por ejemplo, algunas aplicaciones interactivas requieren interacción significativa con el servidor. Las limitaciones en el ancho de banda de las conexiones dial-up limitan severamente las conexiones, y los tipos de interactividad que una aplicación puede soportar. Los temas de rendimiento combinados con problemas de confiabilidad (aun los mayores Web sites no son inmunes a las perdidas de comunicación con el servidor ni de la no disponibilidad del servicio) hacen el diseñar aplicaciones para redes privadas una mejor solución, en algunos escenarios[299].

A continuación se muestra un diagrama que muestra el papel de los Web services en el comercio electrónico[300]:

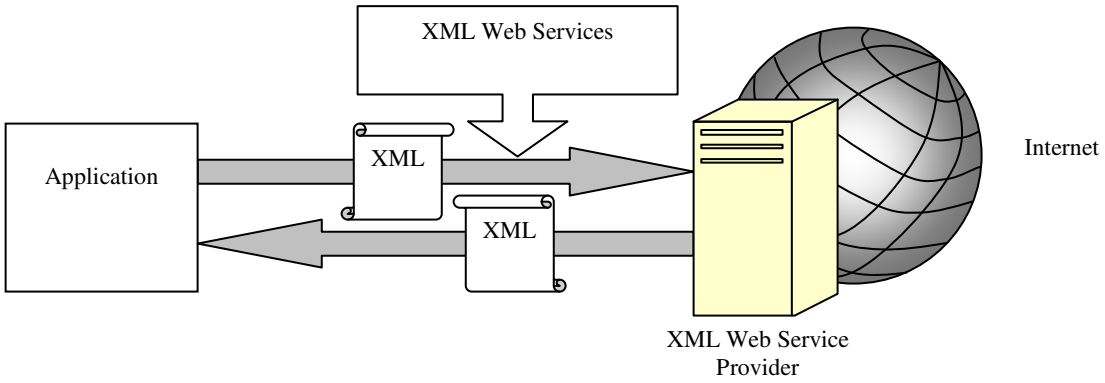


Ilustración 38 Papel de los Web services en el comercio electrónico

[297] Ibid., p. 13
[298] Ibid., p. 12
[299] Ibid., p. 13
[300] MICROSOFT CORPORATION. Introduction to SOAP and ..., p. 3

¿Se tiene una definición formal de los Web services? *Pues bien, un XML Web service es un direccionable conjunto de aplicaciones que está expuesto sobre una red para servir como bloque de construcción en la creación de aplicaciones distribuidas. Similar a la programación por componentes, los XML Web services son "cajas negras"*. Esto es, encapsulan la implementación y proveen una interfaz para comunicarse con el XML Web service Un ejemplo temprano de los Web services es Microsoft Passport. Passport provee servicios de identificación, y es enteramente funcional y utilizable a través de solicitudes de Hypertext Transfer Protocol (HTTP)[301].

Los fundamentos de los XML Web services son HTTP, XML Schemas, WSDL y SOAP. El desarrollo de esas tecnologías está gobernado por World Wide Web Consortium (W3C)

No existe restricción en el tamaño de un XML Web service. Puede ir desde componentes simples como un componente de rastreo de órdenes publicado por una compañía de embarques, hasta grandes aplicaciones financieras en Internet. Se pueden aplicar los XML Web services como diferentes niveles de solución. En este sentido, los XML Web services pueden proveer acceso conveniente a un conjunto estático de información. Por ejemplo un XML Web service puede permitir a un cliente solicitar información demográfica para una ciudad específica de su país.

Alternativamente, los desarrolladores podrían usar un XML Web service para crear aplicaciones altamente interactivas. Como ejemplo, una agencia de viajes podría permitir a través de su portal electrónico construir un itinerario de vacaciones completo en línea utilizando múltiples XML Web services. El usuario podría usar un XML Web service para hacer reservaciones de hotel y renta de auto, planear itinerarios de vuelo, reservar y hasta comprar boletos de avión.Se ha dicho que los Web services son como bloques de construcción de una aplicación lógica mayor. ¿Por qué se requiere en algunos casos programarlos de esta forma? La razón es que un XML Web service puede sumarse a otros XML Web services para proveer un sofisticado conjunto de servicios. Por ejemplo, el XML Web service de un punto de venta podría hacer uso de un XML Web service para verificación de crédito para facilitar la aprobación en línea de aplicaciones de préstamos. En el futuro, un creciente número de aplicaciones distribuidas serán construidas sobre XML Web services. En tales aplicaciones, los XML Web services serán siempre seleccionados en tiempo de ejecución basados en diferentes métricas, tales como disponibilidad, costo, rendimiento y calidad. Este nivel de selección será invaluable en el diseño de sistemas redundantes con las capacidades de falla[302].

¿Porqué utilizar la arquitectura de Web services? Ya se ha hablado de muchas características ventajosas de los XML Web services en la **sección 2.5** pero vale la pena recordar que debido a que los Web services promueven interoperabilidad entre diferentes plataformas de aplicación, se pueden utilizar en un amplio rango de escenarios. Además[303]:

- **Los Web services utilizan estándares como HTTP y XML para interoperabilidad**. Los XML Web services son diseñados para usarse con SOAP. Debido a que SOAP está basado en XML y es de plataforma neutral, los desarrolladores no necesitan averiguar como construir puentes entre plataformas de aplicación,omo lo tiene que hacer Distributed Component Object Model (DCOM), Common ObjectREquest Broker Architecture (CORBA), y otros diferentes protocolos. Algún XML Web service puede interoperar con otro XML Web service diferente.
- **Uso de estándares soportados por la industria**. Los mayores vendedores ahora soportan tecnologías relacionadas con XML Web services, específicamente HTTP, XML y SOAP. El soporte universal de estos estándares no tiene precedente. Este tipo desoporte hace posible y simple comunicar sistemas heterogéneos.
- **Los Web services pueden ser escritos en cualquier lenguaje**. Los desarrolladores pueden escribir XML Web services en el lenguaje de su preferencia. Consecuentemente, los desarrolladores no necesitan aprender nuevos lenguajes o estandarisarse en un solo lenguaje de programación para crear o consumir los XML Web services.
- **Las aplicaciones existentes pueden exponerse como Web services**. Es muy facil exponer componentes y librerías existentes como un XML Web service. Vendedorescomo Microsoft ya proveen herramientas para hacer la tarea de exponer componentes ylibrerías de esta forma. La mayoría de las compañias tienen un gran número de componentes refinados a través de los años, y librerías y aplicaciones que son clave en la definición de reglas de negocios y transacciones en bases de datos. Es más barato reutilizar la funcionalidad de esos recursos de software que rediseñarlos y programarlos en los nuevos lenguajes.

[301] Ibid., p. 3
[302] Ibid., p. 4
[303] Ibid., p. 5

- **Los Application Service Providers pueden almacenar Web services basados en suscripciones**. Los web services atraen ingresos a las compañias a través de renta de tiempo de procesamiento, y renta de información en bases de datos, para aplicaciones de telemarketing y mercadotecnia.
- **Los web services promueven integración de aplicaciones empresariales a través de diferentes plataformas**. Los escenarios para integración de aplicaciones son generalmente caracterizados por una fragmetación de elementos publicados, cuya comunicación es esencial en la coordinación de procesos productivos. En este sentido, cada web service puede proveer un contrato electrónico de uso en formato WSDL que es una gramática particular de XML.

Existe un determinado número de escenarios en el que los XML Web services son la solución apropiada.

I. Application Service Providers/ Hosted Applications B2B
II. Application Integration

Considerando el escenario I, se encuentra que existen proveedores que almacenan aplicaciones y a su vez rentan servicios a los suscriptores finales. Desde la perspectiva del suscriptor, estas son las características de las aplicaciones almacenadas:

- Las aplicaciones que rentan los proveedores son vistas como un portal.
- Las aplicaciones que guardan los proveedores existen en un ambiente aislado.
- Cada suscriptor tiene su propia instancia de la aplicación.
- Los suscriptores no comparten datos con otros suscriptores.

Desde la perspectiva de los Application Service Providers, todas las aplicaciones almacenadas deben alcanzar al menos el siguiente criterio:

- Las instancias de aplicación deben ser configurables por separado para cada suscriptor, aun en un hardware compartido. Esto incluye características de seguridad.
- Las aplicaciones deben tener mecanismos para medir la duración de uso de una aplicación, para propósitos de cobranza.

Esto es útil si en ambos casos un proveedor y una aplicación ofrecen interfases estándar para interacción mutua. Los proveedores de aplicaciones no tienen razón alguna para almacenar sus aplicaciones en sus propias instalaciones físicas/geográficas. En tales casos, el servidor físico de la aplicación debe encontrarse con el proveedor de servidores.

La siguiente figura ilustra el concepto[304].

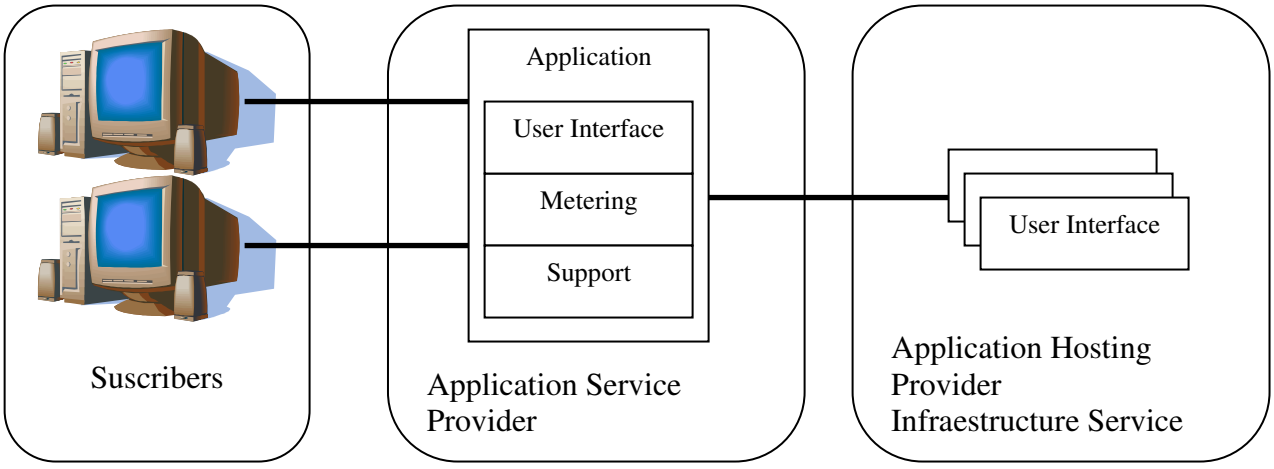


Ilustración 39 Interfases estándar para interacción entre diversas capas de la aplicación

[304] MICROSOFT CORPORATION. The Need for XML Web Services..., p. 20

¿De qué forma trabajan los XML Web services en soluciones Business-To-Business? Los XML Web services son ideales para usar en las soluciones de comercio electrónico negocio a negocio, porque permiten que las aplicaciones soliciten información de forma dinámica desde otras aplicaciones a través de Internet.

En el comercio negocio a negocio, usualmente existe una necesidad de información dinámica en tiemo real, tal como revisiones de crédito, y cálculos de cargos de impuestos y envios, conversiones de tipo de moneda, y actualizaciones de inventario. Puede ser extremadamente caro para una organización proveer cada componente de la transacción distribuida, por lo que los XML Web services son utilizados para permitir que múltiples organizaciones se suscriban a servicios de aplicación, ofrecidos por un administrador central (broker).

El orden lógico del escenario B2B o e-commerce de los Web services es como sigue[305]:

1. Una aplicación llama a los XML Web services.
2. Los XML Web services realizan los cálculos solicitados
3. Los XML Web services envían los resultados calculados a la aplicación

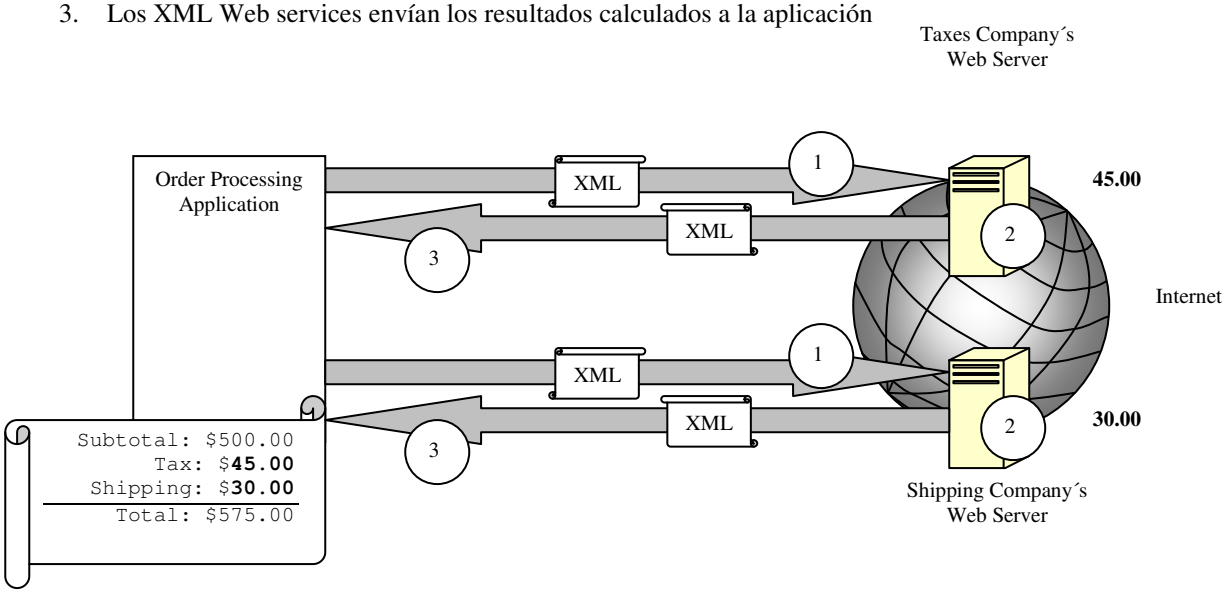


Ilustración 40 Escenario Business to Business o e-commerce de los Web services

¿Qué información aparece en la figura anterior? Allí se muestra una transacción típica Business-To-Business. Se parte del hecho que podría ser útil para una compañía usar un web service de terceros para obtener tarifas actualizadas de impuestos o embarques. Por ejemplo, después que un cliente ha enviado una orden a través de una interfaz Web estándar, la aplicación Web puede enviar el subtotal de la orden a un XML Web service ofrecido por un proveedor de servicios de aplicación. Este Web service puede calcular el impuesto y el embarque basándose en la información más actualizada, provista por aplicaciones de monitoreo dedicadas.

En el punto II de la lista de escenarios de Web services aparece la integración de aplicaciones. Los escenarios para integración de aplicaciones son generalmente caracterizados por la flexibilidad de un contrato de comunicación publicado entre varias aplicaciones que requieren ser integradas.

Los XML Web services proveen útiles capacidades en dichos aspectos. Por diseño, los XML Web services son direccionables por URL, lo cual permite la portabilidad de los componentes. Por otra parte, al usar Web Services Description Language (WSDL) los XML Web services individuales pueden proveer un contrato que describe la interfaz de los XML Web services[306].

[305] MICROSOFT CORPORATION. Introduction to SOAP and ..., p. 7
[306] MICROSOFT CORPORATION. The Need for XML Web Services..., p. 21

Los Web services no solamente pueden ser clasificados por sus escenarios (Application Service Providers contra Application Integration) también pueden clasificarse por su tipo de arquitectura en síncronos y los asíncronos. Los Web services síncronos se amoldan al concepto de RPC. De hecho, las arquitecturas basadas en RPC están entre los primeros candidatos a ser considerados como solución a los problemas de diseño de las aplicaciones distribuidas.

Una llamada a procedimiento remoto (RPC) es una llamada hecha a un procedimiento o función que reside en un sistema remoto. Un RPC se observa como una llamada a procedimientos ordinaria o a una llamada a función dentro del código que lo utiliza. Un RPC provee tanto:

- Transparencia en localización física (el programador no necesita saber la localidad física del proveedor de servicio)
- Un modelo de programación con el que se está familiarizado (la mayoría de los programadores están muy acostumbrados a usar algunas formas de llamadas a procedimiento)

La infraestructura de RPC genera un "stub", el cual actúa como representante del código del procedimiento remoto y ensambla cualquier argumento de procedimiento en el buffer, el cual será transmitido sobre la red al servidor RPC. En el servidor RPC, el "stub" desempaqueta los argumentos en el proceso del servidor y los argumentos se transmiten a la función actual que está siendo llamada. Algún valor de retorno se regresa al que origina la llamada en una forma similar[307].

RPC realiza llamadas a funciones de manera síncrona. En un modelo RPC, una aplicación establece una conversación con el servidor RPC apropiado. Las llamadas a funciones RPC se ven muy parecidas a las llamadas a procedimiento locales; también, las semánticas de bloqueo de RPCs son las mismas que aquellas de llamadas a procedimientos locales. El hecho de que las semánticas de bloqueo sean las mismas significa que las llamadas son síncronas, esto es, el hilo de ejecución se bloquea hasta que la función regresa. Para la mayoría de los desarrolladores, este es un modelo de programación muy cómodo. Sin embargo, colocar un modelo síncrono sobre una arquitectura distribuida llega a introducir algunos problemas[308].

¿Qué situaciones deben manejar las aplicaciones basadas en RPC? Se pueden citar básicamente cinco[309]:

1. **Redundancia en la construcción de módulos**. El primer problema inherente en las arquitecturas basadas en RPC es la localización del servicio. ¿Cómo puede la aplicación descubrir la información necesaria para conectarse a un punto que podría proveer los servicios requeridos? La solución simple utilizada en la mayoría de las aplicaciones es incluir la información del punto final en el código compilado. Esto desde luego no es la solución óptima porque genera redundancia en las instancias compiladas y las capacidades de tratamiento de errores dentro de la aplicación se tornan difíciles.
2. **Disponibilidad de agregación**. En la medida que una aplicación comienza a apoyarse en múltiples servicios distribuidos, llega a ser más susceptible a la posibilidad de que algunos servicios críticos lleguen a no estar disponibles. No obstante, la disponibilidad de agregación de una aplicación distribuida podría ser afectada negativamente por la fragilidad de una implementación típica. Las implementaciones típicas son débiles porque no toleran cambios en sus ambientes de desarrollo, sin que antes ocurran fallas.
3. **Balanceos en la carga de trabajo y tratamiento de errores**. Compilar la información de los puntos finales tales como dirección y puerto en una aplicación resultan en otro problema. Específicamente, no hay una forma simple para una aplicación basada en RPC en que realice alguna forma de balanceo de cargas dinámico. Tampoco puede la aplicación reaccionar a una pérdida del servidor al reintentar alguna operación en un servidor alterno.
4. **Asignación de prioridades**. La asignación de prioridades de solicitudes es casi imposible, debido a que todas las peticiones por default son manejadas en la premisa de primero-entra primero-sale. Si un servidor en particular está pesadamente cargado, los clientes de alta prioridad podrían ser sometidos a retardos inaceptables.
5. **Sobrecarga de trabajo**. Otro problema significativo en las aplicaciones basadas en RPC es la imposibilidad de manejar picos de carga de trabajo. Los picos de carga de trabajo pueden tener las siguientes consecuencias:
   - **Interrupciones** en el funcionamiento del servidor debido a fallas en el mismo.

[307] Ibid., p. 7
[308] Ibid.
[309] Ibid., p. 8

- **Fallas** en las acciones debido a que un servicio solicitado ha sido completamente utilizado (por ejemplo, conexiones en la base de datos)
- La necesidad de mayor hardware del que se requiere con cargas típicas, para manejar los **picos de carga** de datos infrecuentes la mayoría de las veces.

Por otra parte, se tienen los Web services asíncronos. De hecho, **las arquitecturas y soluciones basadas en apilamiento de mensajes son típicamente asíncronas**. El software intermedio orientado a mensajes le provee a la aplicación servicios de comunicaciones interprocesos al usar la tecnología de apilamiento de mensajes como la base para garantizar el nivel del servicio para aplicaciones críticas. La tecnología de apilamiento de mensajes rastrea un mensaje a lo largo de cada rama de su ruta, de la misma forma en que los servicios de mensajería de alguna compañía comercial realizan el rastreo de paquetes. Dicha tecnología para formar los mensajes en fila asegura que algún problema puede ser detectado y aún corregido posiblemente, sin la intervención del usuario.

Las arquitecturas basadas en mensajes han sido usualmente construidas sobre productos de apilamiento de mensajes, tales como Microsoft Message Queuing (antes conocido como MSMQ)[310].

¿Qué características tiene la mensajería asíncrona? Las características más evidentes de las arquitecturas basadas en mensajes son:

- Las arquitecturas son asíncronas
- Están basadas en el intercambio de mensajes en lugar de llamadas a funciones.

Ambas características tienen algunas ventajas, tales como:

- Los mensajes pueden ser dirigidos basados en carga de trabajo y prioridad.
- Las llamadas asíncronas permiten a los clientes realizar trabajo productivo mientras esperan el resultado de una operación que en otro escenario consumiría tiempo[311].

Así como las comunicaciones síncronas tienen limitantes, la mensajería asíncrona también tiene al menos tres restricciones de aplicación[312]:

- **Procesamiento de la carga de datos del mensaje**. Debido a que los sistemas asíncronos son basados en transferencia de mensajes, una de las primeras tareas que el programador de aplicaciones debe tener en cuenta es añadir funcionalidad al empaquetado y desempaquetado del contenido de los mensajes. Después de desempaquetar el contenido del mensaje, *la aplicación debe todavía valida su contenido*. En la medida que la complejidad y la flexibilidad de la carga del mensaje se incrementa, desempaquetar y validar los mensajes podría tornarse difícil.
- **Interoperabilidad**. La mayoría de los sistemas basados en mensajes son programados usando productos propietarios de apilamiento de mensajes. Usar productos propietarios de "message queuing" tiene al menos dos requerimientos en el diseño de sistemas basados en mensaje interoperables. Todas las organizaciones que participan en la operación distribuida deben tener:
  - Programas para apilamiento de mensajes (**message queuing software**)
  - Software para crear un **puente** a fin de operar entre diferentes ambientes de mensajería.

  Aún si los requerimientos anteriores son alcanzables, la solución resultante tiende a ser difícil de llevar a cabo, y cara. Sin embargo, soluciones basadas en mensajería no son viables como una forma estándar de implementar aplicaciones distribuidas.
- **Flujos de datos en línea de trabajo y secuencia de mensajes**. Muchos escenarios de aplicaciones distribuidas envuelven flujos de trabajo que están definidos como una secuencia de mensajes siendo intercambiados entre múltiples computadoras. Debido a que los mensajes son enviados asíncronamente, es posible que los mensajes lleguen sin un orden lógico. En algunos escenarios, esto podría ser fatal si los mensajes son procesados en la secuencia incorrecta. Por ejemplo, si un administrador de inventario recibe ordenes de compra y venta fuera de

[310] Ibid., p. 9
[311] Ibid.
[312] Ibid., pp. 9, 10

secuencia, esto podría afectar significativamente los precios pagados en cada transacción. Esto significa que el desarrollador de aplicaciones tiene la inconveniencia adicional de crear una capa de protocolo de alto nivel sobre el protocolo de mensajes para rastrear la secuencia de los mensajes.

Tanto las arquitecturas basadas en RPC como las basadas en mensajes han sido exitosamente implementadas por varias organizaciones, pero estas arquitecturas sufren determinado número de problemas. En resumidas cuentas, los problemas inherentes a estos escenarios y modelos de objetos distribuidos son tratados con los estándares de Web, los cuales alivian muchos de estos problemas.

Sea que los Web services estén **clasificados por su escenario de aplicación o por los aspectos técnicos de su arquitectura**, siguen siendo soluciones orientadas a servicio. Para construir aplicaciones distribuidas flexibles y robustas, se deben cumplir determinado número de requerimientos básicos.

- Cuando se integran recursos de software, los recursos deben ser flexibles; es decir, deben ser distintos y estar separados.
- La comunicación interprograma debe cumplir con los estándares de Internet.
- Las interfases de servicios para los recursos de software deben ser publicadas para su uso, y las definiciones de interfases y documentación deben ser accesibles.

Construir aplicaciones que alcancen los previos requerimientos resultará en las siguientes ventajas:

- Se podrán construir aplicaciones al integrar procesos de negocios centrales con programas ofrecidos por proveedores.
- Se pueden crear más recursos de software de menor tamaño.
- Recursos reutilizables de terceros pueden proveer beneficios en costo y productividad.
- La venta de software como servicios puede llegar a ser un estándar. Por ejemplo, una compañía podría vender un servicio de calendario compartido como servicio accesible de Web, en lugar de vender una aplicación de calendario tipo "stand alone"[313].

La arquitectura orientada a servicios es ideal para desarrollar aplicaciones distribuidas. Es una arquitectura conceptual para implementar aplicaciones dinámicas, separadas y distribuidas.

Al dia de hoy, la mayoría de los sistemas de negocios y aplicaciones son construidos en aplicaciones y subsistemas fuertemente relacionados. Cuando las aplicaciones y subsistemas son unidas estrechamente, un cambio en un subsistema puede causar que fallen muchos componentes o aplicaciones dependientes entre sí. Esta debilidad de los sistemas existentes es una de las razones primarias por las cuales es de alto costo mantenerlas; esto también limita que tan fácilmente la aplicación puede ser modificada para satisfacer los requerimientos de negocios, aceleradamente cambiantes.

Una arquitectura orientada a servicio consiste de tres roles primarios: service provider, service consumer y service broker. Un diagrama de esta arquitectura se muestra a continuación.

- **Service provider**. Un proveedor de servicios es un nodo en la red (intranet o Internet) que provee acceso a la interfaz del servicio de software que realiza un conjunto específico de operaciones. Un nodo proveedor de servicios ofrece acceso a los servicios de un sistema de negocios, un subsistema o un componente.
- **Service consumer**. Un consumidor de servicios es un nodo en la red que liga el servicio de un proveedor de servicios y utiliza dicho servicio para crear una solución de negocios. En el modelo de arquitectura orientada a servicios, los consumidores de servicios no son aplicaciones, sino nodos. Sin embargo, para motivos prácticos se puede ver un consumidor de servicios como una aplicación cliente en un nodo.
- **Service broker**. Un service broker es un nodo en la red que es un repositorio de descripciones de servicio y puede ser utilizado como una libreta de direcciones para buscar la dirección del servicio. Los consumidores de servicios pueden interrogar al service broker -integrador- para localizar un proveedor de servicios requerido y un servicio.

[313] Ibid., p. 2

Los service brokers usualmente actuan como proveedores de servicios en los casos donde el servicio requerido es "service brokering"[314].

Se puede trazar un diagrama que ilustra las relaciones entre estos tres elementos:

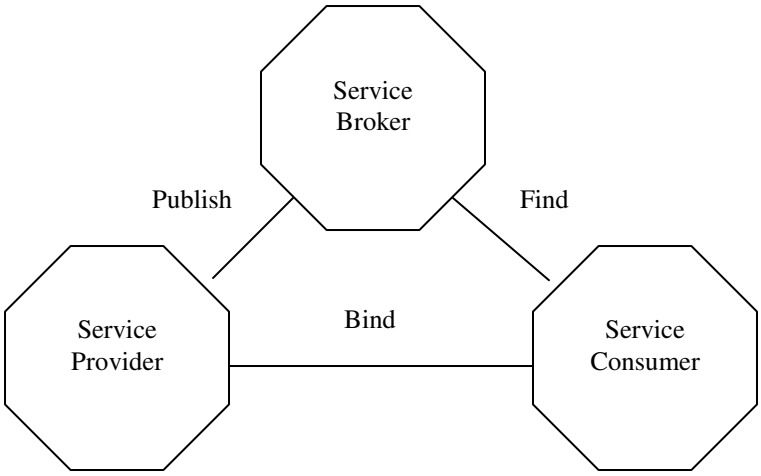


Ilustración 41 Roles primarios de una arquitectura orientada a servicio

¿Qué interacciones entre los elementos de la arquitectura orientada a servicios están descritas en la imagen anteior? Los tres roles de la arquitectura orientada a servicio interactuan para realizar tres operaciones básicas:

- **Servicios de publicación**. Los proveedores de servicios publican sus servicios en un Service Broker. La información publicada incluye la definición de interfase del servicio, localización de proveedores de servicio, y posiblemente otra información de soporte o documentación relacionada.
- **Servicios de búsqueda**. Los consumidores de servicios buscan los servicios requeridos/deseados al usar el service broker.
- **Ligas a servicios**. Los consumidores de servicios ligan servicios específicos que son provistos por un Service Provider. El proceso de ligado incluye identificación de los consumidores. La seguridad está integrada en el usuario/password.

En ambos casos, buscar y ligar servicios puede realizarse dinámicamente para permitir que las aplicaciones se configuren por sí mismas dinámicamente. Por ejemplo, si una aplicación encuentra que el tiempo de respuesta de un proveedor de servicios llega a ser inaceptable, entonces podría decidir cambiar a otro proveedor de servicios en tiempo de ejecución[315].

Los elementos del diagrama de la arquitectura orientada a servicios hacen recordar los elementos básicos en una arquitectura de XML Web Services Dichos son[316]:

- **El proveedor de XML Web services**, quien es un nodo de red que almacena un XML Web service.
- **El consumidor de XML Web services**, el cual es un nodo de red que almacena algún cliente que puede comunicarse utilizando Hypertext Transfer Protocol (http). Los clientes incluyen browsers, aplicaciones de consola, y aplicaciones con interfases gráficas de usuario tradicionales.
- **El XML Web service broker**, el cual es un nodo de red que almacena un registro global de XML Web services disponibles, muy similar al libro de direcciones global.

[314] MICROSOFT CORPORATION. "XML Web Service Architectures" En: Developing XML Web Services Using Microsoft© ASP.NET, p. 3
[315] Ibid.
[316] Ibid., p. 5

Todos estos nodos de red deben tener la habilidad de comunicarse con otros típicamente a través de una red en Transmission Control Protocol/Internet Protocol (TCP/IP).

El diagrama siguiente muestra las relaciones entre varios elementos de la arquitectura de XML Web services.

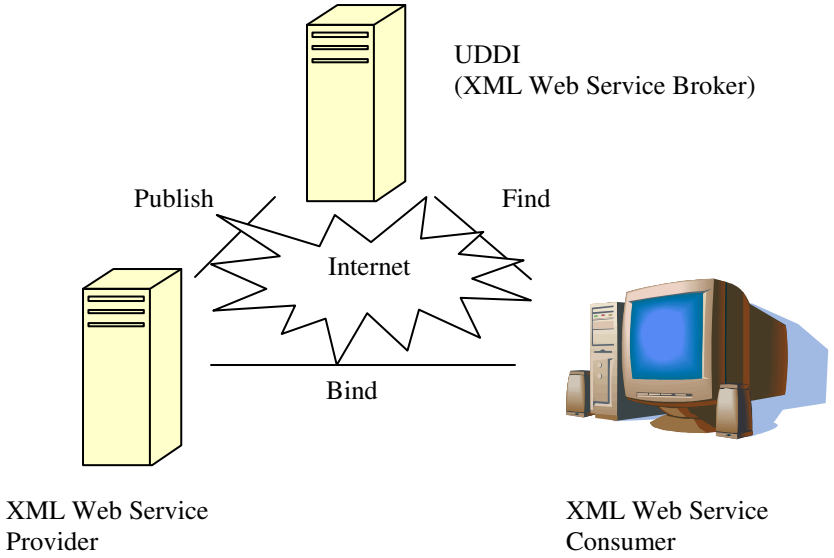


Ilustración 42 Arquitectura de Web Services

Al ver la correspondencia exacta entre los elementos de la arquitectura orientada a servicios y la arquitectura de XML Web services, puede decirse con confianza que los Web services son un caso particular de la arquitectura orientada a servicios. En una solución basada en XML Web services, *los tres nodos definidos en una arquitectura orientada a servicios corresponden* a los elementos de la solución en XML Web Services.

- **El service broker en los XML Web services**. El papel del service broker es representado por un nodo que almacena el Universal Description, Discovery and Integration Registry (UDDI).
- **El proveedor de servicio en los XML Web services**. El papel del proveedor de servicios es representado por el nodo que expone los XML Web services a través de páginas como ASP.NET con la extensión .asmx. En el caso de Java, el proveedor de servicios es Tomcat con servlets y Java Server Pages.
- **El consumidor de servicios en los XML Web services**. El papel de un consumidor de servicios es cubierto por algún nodo que pueda comunicarse utilizando el Simple Object Access Protocol (SOAP) o bien http, comprenda la interfaz del servicio que está siendo utilizada, y pueda proporcionar las credenciales de identificación necesarias.

El siguiente diagrama ilustra mejor estas relaciones[317].

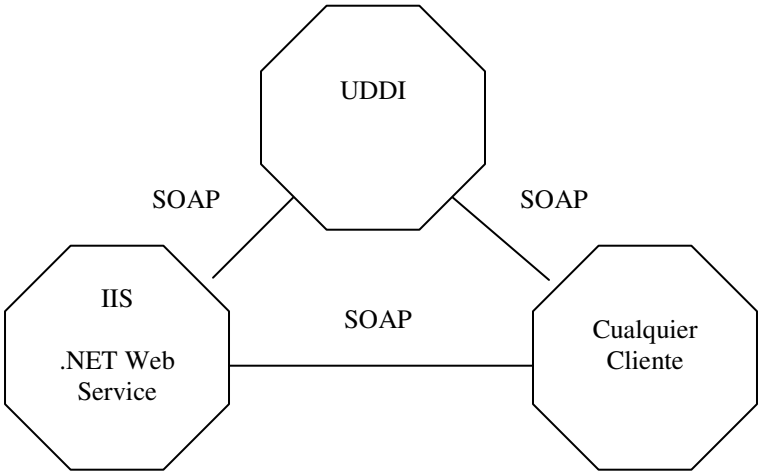


Ilustración 43 Los Web services son un caso particular de la arquitectura orientada a servicios

Los Web servers son los elementos básicos dentro de un XML Web Service Provider. Como mínimo, un proveedor de XML Web services debe incluir un elemento que pueda hablar y escuchar el protocolo de red. Para los XML Web services que han sido desarrollados usando Microsoft .NET Framework y Microsoft Visual Studio .NET, el protocol listener debe ser un http listener.

Debido a que el proveedor de un XML Web service podría estar almacenando múltiples XML Web services, este también debe poder direccionar la solicitud a su XML Web service apropiado. Eso es análogo al servicio Remote Procedure Call Subsystem (RPCSS) que es responsable de manejar solicitudes entrantes de Distributed Component Object Model (DCOM) y dirigirlas al servidor apropiado Component Object Model (COM).

Por causa de que hay potenciales consumidores desconocidos de los métodos de un XML Web service, el Web Server debe estar protegido con servicios de seguridad básica a nivel protocolo. Por ejemplo, Microsoft Internet Information Services (IIS), el cual es un Web Server, provee todos los servicios que los XML Web services requieren a través de sus características:

- IIS es un receptor de HTTP
- IIS actúa como puerta de enlace a la implementación de varios XML Web services que podría almacenar, a través de la arquitectura Internet Server Application Programming Interface (ISAPI)
- IIS provee infraestructura de seguridad significativa[318].

Un servidor Web, tal como IIS, puede invocar un servicio en representación del cliente, utilizar muchas opciones diferentes. Un servidor web puede iniciar una aplicación Common Gateway Interface (CGI); ejecutar un intérprete de scripts como se puede hacer en Microsoft Active Server Pages,o invocar una aplicación ISAPI.

Cuando IIS trabaja en conjunción con el lenguaje común en tiempo de ejecución, utiliza un filtro ISAOU para interceptar solicitudes para páginas con la extensión **`.asmx`** y entonces iniciar un servidor de tiempo de ejecución. El servidor de tiempo de ejecución entonces ejecuta el código para un XML Web service que es implementado al usar .NET Framework.

IIS no está restringido al almacenamiento de XML Web services basados en .NET También puede almacenar XML Web services basados en Microsoft Active Template Library (ATL) los cuales son compilados en Visual C++. Los XML Web

[317] Ibid., p. 6
[318] Ibid., p. 9

services basados en .NET proveen algunas ventajas significativas. Una de las más significativas ventajas es la infraestructura de seguridad flexible que la plataforma .NET provee[319].

¿Qué ejemplos de proveedores de XML Web services existen? Si una organización desea proveer XML Web services, debe ser capaz de proveer algún tipo de servicio electrónico. Debido a que casi cualquier pieza de funcionalidad puede ser clasificada como servicio, es imposible enumerar todos los tipos posibles de XML Web services. Sin embargo, dos ejemplos domunes de proveedores de XML Web serivces son vendedores de software independientes y procesos de negocios de propósito general.

- **Vendedores de software independientes**. Ellos son dueños de productos que realizan una variedad de tareas. Estos productos pueden ser expuestos como XML Web services individuales o XML Web services integrados en una federación. Por ejemplo, una compañía que vende una aplicación de cálculo para impuestos personales podría solicitar hacer dicha aplicación pública, a través de un XML Web service.
- **Procesos de negocios de propósito general**. Dichos están suficientemente generalizados para la adopción de una amplia variedad de clientes, y también pueden ser expuestos como XML Web services. Por ejemplo un servicio de procesamiento de nómnia puede ofrecer sus servicios de administración de nómina como un XML Web service[320].

Ahora bien, ¿Qué conjunto mínimo de *funcionalidades que debe reunir un consumidor* de XML Web services? Para consumir un XML Web service, un consumidor debe llamar a los métodos del XML Web services con los parámetros apropiados al utilizar los protocolos (por ejemplo, SOAP) que el servicio soporta.

Es difícil formatear los mensajes correctamente antes de pasarlos al XML Web services, y es también difícil manejar los detalles de los protocolos que los XML Web services soportan. Los lenguajes de programación modernos, tales como la infraestructura .NET, proveen clases que encapsulan la mayoría de los detalles de bajo nivel. Al estar encapsulados dichos detalles, se libera al desarrollador de tener que implementar dicha infraestructura. Entre esos detalles se encuentran los siguientes:

- **Localización de servicio**. Antes de que un XML Web service pueda ser utilizado, un consumidor debe poder localizarlo. Localizar un XML Web service puede realizarse estáticamente al incluir la información del punto final en el código compilado. Alternativamente a esto, el consumidor de XML Web services puede descubrir de una forma dinámica la localización del XML Web service, en tiempo de ejecución. Esta característica provee al consumidor de XML Web services la flexibilidad de seleccionar entre XML Web services equivalentes y competitivos, basándose sencillamente en el criterio de precio o rendimiento.
  El mecanismo estándar para localizar los XML Web services apropiados, su descripción de servicio, y sus puntos finales es a través de un UDDI registry.
- **Proxies**. Cuando se implementa un consumidor de XML Web services, los desarrolladores pueden invertir tiempo que podría ser más productivo para otras cosas, y deberían no tener nada que ver con las siguientes tareas:
  - Trabajar con los protocolos subyacentes.
  - Interpretar las cadenas de bytes para extraer los datos.
  - Validar las cadenas de datos de entrada.
  - Construir los paquetes de datos de salida.

  Sin embargo, el desarrollador es la mayoría de las veces forzado a manejar dichas tareas debido a que no existe código preconstruido disponible para realizar dichas tareas. Una aproximación típica para manejar esas tareas es encapsular u ocultar los detalles de implementación en una clase envolvente que actúe como "Proxy" para el XML Web service. No solamente las clases Proxy ocultan los detalles de implementación, sino que también provee al desarrollador un modelo de programación familiar para llamar métodos de los objetos.
  El único problema con esta técnica es que una clase Proxy debe ser implementada para cada interfaz de XML Web service con la cual un consumidor de XML Web services desea interactuar.
  Debido a que una interfaz de XML Web services está definida utilizando XML, es adecuado y directo escribir herramientas que puedan generar automáticamente las clases envolventes para el Proxy.

[319] Ibid., p. 10
[320] Ibid.

- **Llamadas asíncronas**. Debido a que los XML Web services son regularmente utilizados sobre redes que no son confiables ni rápidas como las redes de área local (LANs), es usualmente mejor programar consumidores de Web services que realicen llamadas asíncronas a los XML Web services.
- **Los datos consultados en un XML Web service** deben tener la capacidad de ser formateados, filtrados, catalogados y disponibles de búsqueda de acuerdo a las preferencias del cliente[321].

Finalmente, ¿Qué características debe reunir el XML Web service Broker a fin de interactuar adecuadamente con los primeros dos elementos de la arquitectura? Tal como una arquitectura orientada a servicio necesita de un "service broker", la arquitectura de XML Web services también necesita de un "service broker". Para facilitar dichas interacciones, los negocios necesitan una ámplia solución para publicar su información a algún cliente o socio de negocios alrededor del mundo. Un XML Web service broker interactua tanto con los proveedores de XML Web services como con los clientes para proveer esta funcionalidad como valor agregado.

Los medios comunes de publicar la información acerca de servicios de negocios hacen posible a las organizaciones:

- Descubrir rápidamente los socios de negocios correctos de entre millones que se encuentran en línea.
- Definir cómo conducir los negocios después de que un socio preferencial es descubierto.
- Crear una aproximación de alcance industrial para negocios para integrar rápida y fácilmente a clientes y socios en Internet. Las organizaciones deben estar capacitadas para compartir información sobre sus productos y servicios, para establecer como prefieren ser integradas en sistemas y procesos de negocios ajenos[322].

En este punto debe hacerse referencia a que no basta con enumerar y describir exhaustivamente cada uno de los elementos en la arquitectura orientada a servicios. Se requiere conocer:

a. **Las interacciones entre brokers y providers**. Los brokers especifican a los proveedores que tipos de información necesitan hacer pública, y entonces publicar dicha información. Los tipos de información publicada por un broker incluye:

   - Clasificar la información para permitir que los XML Web services estén por categoría.
   - Contactar información para el XML Web service.
   - Una descripción textual de los servicios ofrecidos.
   - Llamadas a documentos, proveyendo información acerca de los XML Web services que el proveedor almacene.
   - La localización de los puntos finales de los XML Web services.

   Estas localidades son normalmente almacenadas como Uniform Resource Locators (URLs) que muestra la localización del XML Web service publicado. Debido a que no es posible especificar toda la información en el repositorio del broker, deben existir apuntadores a URL o recursos basados en archivo que facilitarán los descubrimientos. Por ejemplo. El niel de servicio garantiza o informa tomando en cuenta los requerimientos de identificación que pueden ser descubiertos solo en la localidad del XML Web service provider.

b. **Las interacciones entre brokers y consumidores.** La interacción primaria entre consumidores de XML Web services y los brokers relacionados está relacionada con la búsqueda. Los brokers deben facilitar la búsqueda de su repositorio para permitir que los consumidores de XML Web services localizen los servicios apropiados y entonces descubrir la información que es requerida para enlazarse a dicho XML Web service[323].

c. **Necesidad de la definir un broker**. Existen muchas aproximaciones para proveer los servicios de broker en un XML Web service.

[321] Ibid., pp. 11, 12
[322] Ibid., p. 13
[323] Ibid., p. 14

Una aproximación simple es tener toda la información de los socios de comercio potenciales entre sí, utilizando por ejemplo un método específico creado para dicho propósito. En esta aproximación, no se requiere específicamente un broker. Por ejemplo, algunas organizaciones utilizan el método simple de *Electronic Data Interchange* (**EDI**), para publicar una lista de las especificaciones de documento EDO requeridas, que los socios de comercio deben utilizar en un Web site. El problema con esta aproximación es que no existe una forma fácil de descubrir cual de los negocios externos son compatibles con nuestro negocio.

Otra aproximación es tener todos los negocios publicados en un archivo XML Web services description, en nuestro Web site. Entonces, los analistas de Web pueden ingresar automáticamente a un URL registrado y pueden utilizar el índice de los archivos de descripción para los XML Web services que son encontrados en cada Web site. Un XML Web service broker puede entonces proveer un portarl que provea acceso a los índices que los analistas de red construyen. Confiando en que los analistas de Web proveen índices para los XML Web services tiene problemas similares a los problemas encontrados hoy con las máquinas de búsqueda en el Web estándaresy los catálogos que tenemos hoy. El problema es que no hay un mecanismo para asegurar la consistencia en los formatos de descripción de servicio y para la búsqueda fácil de cambios en donde estos ocurran. Tal como las máquinas de búsqueda en el Web regresan muchas ligas no válidas, tales mecanismos de XML Web services pueden también resultar en descripciones de servicio e información de ligado de servicio caducadas.

La aproximación de brokering que ha sido escogida para utilizarse en los XML Web services descansa en un registro distribuido de negocios y sus descripciones de servicio que están programadas en un formato común de XML. La solución que hace posible esta aproximación para resolver el problema de descubrimiento es conocida como Universal Description, Discovery and Integration (UDDI)

Con respecto al punto (c) de la lista anterior, se ha desarrollado previamente el tema de UDDI en la **sección 3.3**, de la cual solo se recordará que UDDI es una especificación para información distribuidas basada en Web acera de registros de XML Web services. UDDI es también un conjunto público de desarrollos de la especificación que permite a los negocios registrar información acerca de los XML Web services, que ellos pueden ofrecer, por lo que otros negocios pueden encontrarlos.

El componente central de un registro que cumple con especificaciones UDDI es un elemento de registro de negocios, el cual es un archivo XML que describe una entidad de negocios y sus XML Web services. Conceptualmente, la información especificada en un registro de negocios tiene tres partes:

- Direcciones de negocios, información de contactos, identificadores conocidos, y así sucesivamente.
- Listas de categorías industriales que están vasadas en taxonomías estándar.
- La información técnica acerca de los XML Web services que el negocio expone. Esta información incluye referencias a la interfaz de especificación en XML Web service, y potencialmente, apuntadores a varios mecanismos de descubrimiento basados en archivo y URL[324].

En conclusión, ahora se sabe porqué es necesario conocer todos estos detalles de las tecnologías de Web services. La razón es que **para programar o consumir exitosamente un XML Web service**, es importante *comprender las características clave del modelo de programación de los XML Web services*. Es también importante comprender algunas de las ramificaciones del modelo de programación. Esas características clave son:

- **Protocolos de Web**. La primer característica del modelo de programación de los XML Web services es que el protocolo de comunicaciones es típicamente http. Sin embargo, http no soporta intrínsecamente el concepto de llamada a método. Debido a esta restricción, los consumidores de XML usualmente utilizan SOAP basado en XML sobre http para invocar a los métodos del XML web service. Sin embargo, es esencial para un desarrollador tener al menos un conocimiento de trabajo de http y SOAP.
- **Carencia de estado**. La mayoría de los desarrolladores están familiarizados con el modelo de objetos basado en "estado de la aplicación". En otras palabras, una instancia de clase es creada y entonces varias operaciones son

[324] Ibid., p. 15

realizadas en el objeto. Entre cada invocación de método, el objeto mantiene su estado. En un ambiente sin estado, el objeto no retiene estados entre las llamadas. Algún estado que necesite ser persistente entre las llamadas puede cer almacenado en una base de datos o en una "cookie".

Los XML Web services no son objetos en el sentido tradicional. Cuando se utiliza ASP.NET para programar un XML Web service, se puede utilizar una clase de C# para diseñarlo. Esta clase está descrita en una página ASP.NET con la extensión .asmx. Cuando la página es procesada, una instancia de esta clase es creada. El tiempo de vida de la página .asmx liga el tiempo de vida del objeto resultante, lo cual significa que una instancia de objeto puede manejar cada invocación de método. Como resultado, las clases que implementan un XML Web service son sin estado. Aunque diseñar sistemas sin estado puede ser inicialmente más difícil, dichos sistemas pueden hacerse más grandes en la medida que la carga de trabajo en dicho sistema se incrementa.

- **Distribución de objetos**. En una aplicación no distribuida, si alguno de los recursos de software requeridos, tales como librerías de funciones en una dynamic link library (DLL), están disponibles al momento que la aplicación es llamada, estos recursos estaran disponibles durante el tiempo de vida de la aplicación. Usualmente, dichos estarán disponibles en cada uso sucesivo de la aplicación. Para aplicaciones distribuidas, especialmente aquellas que hacen uso de recursos de software sobre Internet, existe una disponibilidad incrementada que es requerida para con los recursos de sofware, aunque estos no estén siempre disponibles. Por tanto, las aplicaciones distribuidas que estan construidas con los XML Web services deben ser más conscientes de que los recursos de sofware llegan a estar no disponibles, aún en su tiempo de ejecución.

  Como consecuencia, las soluciones basadas en XML Web services deben estar distribuidas, por lo que deben tener la capacidad de configurarse dinámicamente, si un recurso llega a estar no disponible. Esta forma de ver los sistemas tiene la ventaja de permitir la recuperacion de fallas porque los consumidores no tienen preferencias por alguna instancia en particular de un XML Web service.
- **XML as Universal data format**. El formato universal de datos que es utilzado en los XML Web services es XML. Las siguientes son algunas de las áreas en donde XML es utilizado en los XML Web services:
  - El protocolo SOAP es basado en XML
  - Las descripciones de los XML Web services son documentos XML
  - Los datos recibidos de un Web service están dentro de un documento XML.
  - Los XML Web services están registrados con un registro UDD utilizando documentos XML que son descripciones del servicio de negocios.
  - Las aplicaciones ASP.NET están configuradas utilizando archivos de configuración de XML.
  - Además, los datos recibidos por el XML Web service, son validados por un XML Schema. Es decir, se utiliza un XML para validar otro.

El **capítulo 4**, a continuación, mostrará cómo se genera una aplicación de XML Web services, con un objetivo específico, a saber, envío de órdenes de reparación de taller y envío de reportes de ventas de la industria automotriz hacia un corporativo, la cual ilustrará todos los conceptos descritos en el **capítulo 3** del presente trabajo.

4 Base de datos MySQL con interfaz Java Web Services como respuesta al problema

4.1 Hipótesis

¿Qué es una hipótesis? Es el planteamiento anticipado de una conjetura o suposición que se pretende demostrar mediante una investigación. Es una suposición admitida como provisional y que sirve de punto de partida para una investigación científica. Esta demostración se realiza a través de los siguientes pasos:

a. **Planteamiento concreto del problema a resolver**.- Consiste en plantear precisa y completamente el problema que se trata de resolver, la problemática a solucionar, y las opciones supuestas que se hayan identificado de éste.
b. **La suposición que se quiere llegar a demonstrar**.- Es el concepto supuesto que se anticipa y se quiere llegar a comprobar o desaprobar mediante una aplicación de los métodos de investigación elegidos.
c. **Confirmación de los conocimientos y las suposiciones que se presume que sucederán**.

¿De que forma se va a sintetizar una hipótesis? Síntetizar es reunir elementos en un todo. Es hacer un resúmen o compendio. Es también el método de demostración que procede de los principios a las consecuencias, de las causas a los efectos. Algunos han dicho que la síntesis es la operación inversa al análisis. Por otra parte, la síntesis también es exponer un conjunto de elementos y apreciarlos globalmente como un todo. Sintetizar puede ser también componer artificialmente un cuerpo a partir de diversos elementos[325]. De ésta manera, reuniremos los elementos necesarios para demostrar la necesidad de un sistema de Web services en un ambiente distribuido y con necesidades de información.

¿De qué manera se puede explicar el concepto de síntesis aplicado a un trabajo de investigación? Una obra de consulta da cinco aproximaciones sobre cómo construir una síntesis[326]:

- Una síntesis es una **propuesta** de corto o largo plazo acerca de cómo resolver una problemática, o bien cómo demostrar una hipótesis.
- La síntesis es el **proceso** de acercar un modelo ideal con un escenario real, así como plantear, implantar, integrar y evaluar una solución factible, resultado de dicho proceso.
- La síntesis es una **respuesta** concreta a un problema previamente planteado.
- La síntesis es **investigación aplicada** para reconocer una situación difícil que debe ser resuelta, seleccionar y utilizar una técnica para resolverla de entre diversas alternativas y comprobar si realmente la técnica elegida fue la adecuada para hacer la tarea que preocupa al investigador.
- La síntesis es la **aportación a la comunidad científica** que hará el trabajo de investigación sobre un tema específico.

Ahora que se tiene una definición acotada de la síntesis para los trabajos de investigación documental. ¿Es posible dirigirla conforme al enfoque de sistemas? Pues bien, pensar sintéticamente es utilizar el pensamiento de sistemas[327]. Debido a que la interfaz de Web services es un sistema, es necesario buscar una definición formal del término "Sistema". En éste sentido, un sistema es un conjunto de dos o más elementos que exhiben las siguientes propiedades:

- Las propiedades ó el comportamiento de cada elemento del conjunto tiene un efecto en las propiedades o comportamiento del todo.
- Las propiedades ó el comportamiento de cada elemento y la forma en que afectan al todo dependen de las propiedades y comportamiento de al menos otro elemento del conjunto.
- Cada subgrupo exhibe las dos propiedades anteriores.

[325] LUCENA Cayuela., et. al., Diccionario Enciclopédico 2003, pp. 930,931
[326] BAENA, Guillermina. Tesis en 30 días, pp.40-45
[327] FUENTES Zenón, Arturo. El enfoque de sistemas, p.9

Esto implica que el sistema es divisible desde un punto de vista estructural, pero indivisible desde un punto de vista funcional[328].

En el pensamiento sistémico existe la tendencia a ver los sistemas como parte de sistemas mayores. Su método viene apoyado en las doctrinas del expansionismo y la teleología. El expansionismo está sustentado en tres argumentos:

a. El todo que se desea entender es conceptualizado como parte de un todo mayor.
b. Se busca conocer el comportamiento y características del todo mayor.
c. El todo se explica de acuerdo con el papel e influencia que tiene en el todo más ámplio.

La teleología por su parte, trata de explicar el presente en términos del futuro. Eso implica que se debe responder la pregunta ¿Qué efecto se desea producir? a la vez que se evade la tentación de buscar los fenómenos resultantes causa-efecto[329]. Esto significa que no solo se necesita una semilla para tener una planta, sino que se consideran todas las variables que influyen en el marco conceptual, a saber el Método, Maquinaria, Materia Prima y Medio Ambiente.

La conjetura o suposición -hipótesis- que se quiere sintetizar en este trabajo de investigación establece que *es necesario y suficiente diseñar e implantar una interfaz mediadora de los sistemas certificados y los sistemas corporativos, que transmita reportes de ventas y de órdenes de reparación entre estos actores*, para que resuelva la problemática detallada en el **capítulo 1**, en la situación de la industria automotriz.

La investigación aplicada propone nuevos procesos, un nuevo sistema o una nueva interfaz –un elemento tecnológico artificial que redefine las dependencias funcionales del todo- para dar respuesta a cada una de las diez directivas del enunciado en el problema original. Por tanto, "teleológicamente" hablando, el escenario del problema se verá afectado positivamente de la siguiente manera:

1. **La interfaz hará que los sistemas certificados puedan enviar la información "en línea"**. Al momento que se efectue una transacción se creará el movimiento correspondiente en el sistema corporativo usando la interfaz, para evitar que se envíe la información extemporáneamente.
2. **La interfaz tendrá el propósito de evitar la recaptura de información**. El distribuidor no tendrá que capturar la misma información dos veces. La interfaz integrará la información del sistema certificado en el sistema corporativo, debido a que se trata de la misma información. El mecanismo de integración será un sistema de información en Web services.
3. **La interfaz contendrá métodos de validación de información recibida**. La interfaz podrá introducir sistemas de validación basados en Autómatas Finitos Determinísticos, o en XML Schemas. Estos sistemas de validación estarán situados en la fase de recepción de la información para evitar completamente la recepción de información inválida. La interfaz debe validar los datos que toma de los proveedores certificados y entrega al corporativo de la industria automotriz usando un mecanismo flexible y adaptable a los bruscos cambios de las políticas de la organización. En caso de que la interfaz detecte un conjunto de datos no válidos en algún mensaje recibido, desechará dicho mensaje y responderá al distribuidor la causa por la cual desechó el mensaje.
4. **La interfaz cumplirá con las normas de seguridad corporativas**. Esto lo hará desde el momento que utilice protocolos de transporte seguros, basados en texto, tales como HTTP. La interfaz estará situada en una red corporativa, con lineamientos preestablecidos de seguridad y restricciones en transferencia de datos (switches, routers, firewalls, telephony earth stations, personal earth stations, hybrid earth stations, equipo satelital, etc).
5. **La interfaz utilizará datos autodescritos o metadatos**. Esto ayudará a la organización a eliminar los "layouts" por separado que se proveían junto con los archivos ASCII, los cuales no tenían una norma estandarizada de descripción de datos. Se pensó que los datos están perfectamente autodescritos al usar XML para codificarlos, preferentemente usando la gramática particular SOAP.
6. **La interfaz se manufacturará con costos de programación mínimos**.
    - Utilizaremos en nuestro caso sistemas operativos, motores de bases de datos, lenguajes de programación, motores de servidor web y librerías de adquisición gratuitas, en la medida de lo posible.

[328] Ibid., p. 10
[329] Ibid., p. 11

- Vigilaremos también que la interfaz sea la mejor opción en tiempo de programación y adaptación. El tiempo deberá ser reducido, comparado con otras opciones o metodologías. Tiempo también es dinero.

7. **La interfaz trasladará el sistema heterogéneo en un sistema homogéneo**. En vista de que los proveedores certificados de software no desarrollan ni en el mismo sistema operativo ni en el mismo lenguaje de programación, ni con la misma metodología, cada proveedor certificado de servicios se convierte en una variable diferente. Por tanto debe pensarse en una forma de transferencia de información a la compañía que no dependa de dichas variables.
8. **La interfaz deberá manejar no dos, sino cuatro tipos de información**. Adicionalmente a los reportes de ventas y órdenes de reparacíón, la interfaz Java Web services cuidará de proveer metodos para manejar información de inventario de distribuidores, transferencias e intercambios de automóviles entre ellos.
9. **La interfaz será programada utilizando estándares y tecnología de punta**. Los Web services están siendo adoptados rápidamente como el estandar en arquitecturas orientadas a servicio, lo que hace que todos los fabricantes especializen sus herramientas para ésta tarea.
10. **La interfaz aportará un nuevo volumen de datos recibido por el distribuidor.** La interfaz estimulará nuevas prácticas de negocios electrónicos en la red, para tener los datos en línea, y recibirlos al momento de que se generen, y no tiempo después.

¿Que elementos compondrán al sistema? En secciones anteriores quedó claro que:

- **La arquitectura de sistemas distribuidos** que se probará son los XML Web Services de la *W3C*.
- **El lenguaje de programación** que se utilizará es Java 2 Standard Edition, con un componente denominado "Java Web Services Developers Package" parte de Java 2 Enterprise Edition, también propiedad *de Sun Microsystems*, y que contiene el Apache Tomcat Web Server de la *GNU*.
- **Los servicios de bases de datos** que soportarán la infraestructura es MySQL, de *MySQLAB*.
- **La forma en que se validarán la estructura y los datos** en los mensajes recibidos por el servidor serán los XML Schemas de la *W3C*.

Ahora bien, ¿Cuál es la arquitectura propuesta, que según la hipótesis llega a resolver la situación problemática supracitada? En el próximo diagrama se mostrará el lugar que ocupa el cliente, la base de datos y los web services dentro del diagrama de estereotipos simple.

Éste diagrama lo generamos con el propósito de dar a conocer cómo se integrará la información que se recibía por ftp (de órdenes de reparación) y por VT3270 (de reportes de ventas), y de qué manera pretende recibirse por HTTP, con el objetivo de homogeneizar los protocolos de transporte de datos y utilizar un medio de almacenamiento para guardar los mismos.

Desde la base de datos se elaborarán consultas para enviar la información correspondiente al proveedor de encuestas (en el caso de los reportes de servcio) y para las oficinas centrales de Estados Unidos (en el caso de los reportes de ventas). Ésta medida permitirá eliminar el servidor de ftp y el "link de datos" vt3270, el cual abría puertos no deseados en el firewall.

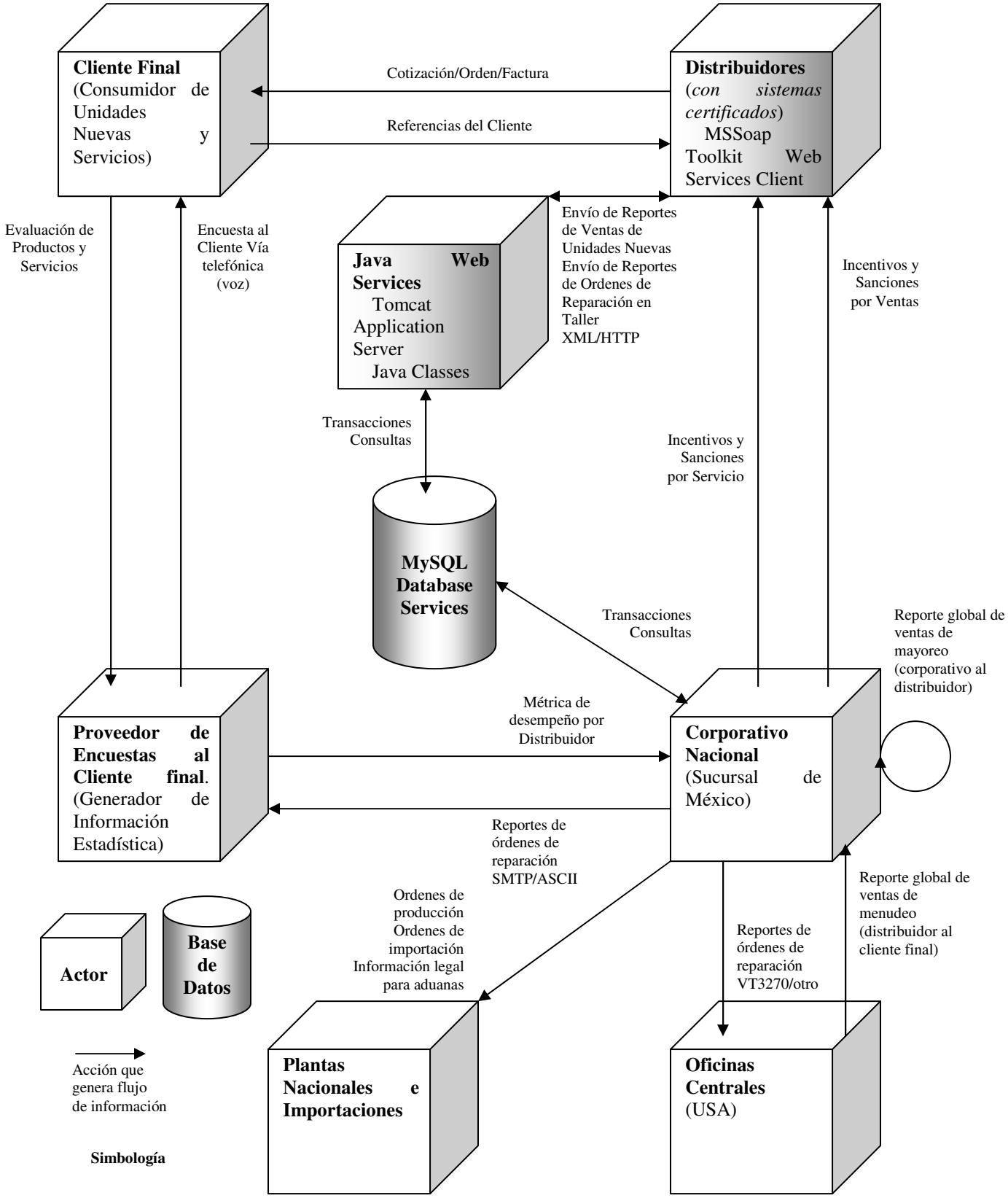


Ilustración 44 Solución propuesta para el caso de estudio

Aislando la aplicación del escenario general, se tienen estos tres actores a desarrollar:

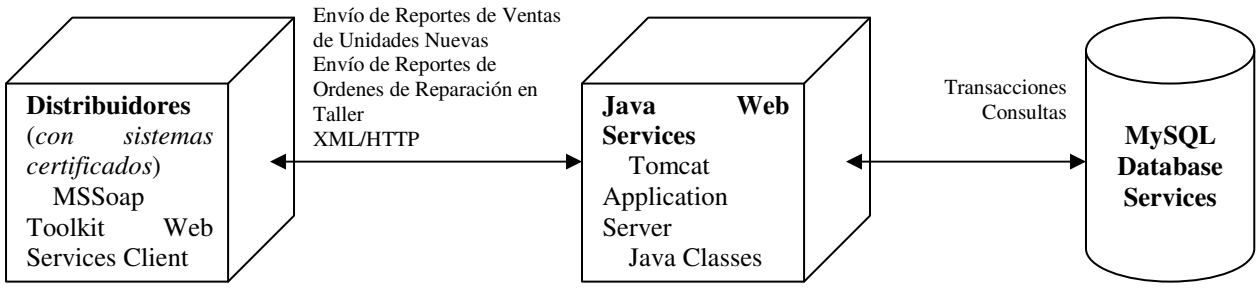


Ilustración 45 Tres componentes en la solución para el caso de estudio

Para analizar ésta solución, las próximas secciones darán detalles técnicos acerca de cómo es:

- …el diseño de la base de datos (**sección 4.2**). Esta sección dará descripción al bloque "**MySQL Database Services**".
- …la creación de la interfaz para efectuar transacciones en el sistema, los métodos para ventas y para servicio (**sección 4.3**). Esta sección dará descripción al bloque "**Java Web Services**".
- …el diseño de las reglas de validación de datos en los XML Schemas (**sección 4.4**). Esta sección también describirá una parte importante del bloque "**Java Web Services**"
- …el diseño y la utilización de la interfaz de Web services usando MSSoap Toolkit (**sección 4.5**). Esta sección describirá los elementos encontrados en el bloque "**Distribuidores**". Aquí se destacará la forma de utilizar los Java Web Services, sin estar obligados a definir a detalle la arquitectura del sistema certificado.

4.2 Diseño de la base de datos

En la definición del problema se definieron dos tipos de información: información para ventas e información para servicio en el corporativo. Al clasificar la información de ventas, se encuentran tres tipos de información, lo que requiere una tabla por cada uno de los tipos de información:

Entidades de Ventas	Movimientos Históricos de Ventas	Catálogos de Ventas
Agencias Clientes Vendedores	Ventas Inventario Transferencias	Tipos de Ventas Tipos de Pagos Estados Clave del Cliente Teléfonos Tipos de Teléfonos

Tabla 23 Información en Ventas

De igual manera, la información de servicio se agrupa en tres tipos de información:

Entidades de Servicio	Movimientos Históricos de Servicio	Catálogos de Servicio
Agencias Clientes Asesores	Ordenes	Estados Tipos de Operación Tipos de Órdenes Tipos de Servicio Clave del Cliente Teléfonos Tipos de Teléfonos

Tabla 24 Información en Servicio

Al hacer un comparativo de las dos tablas, se encuentra que ventas y servicio requieren información acerca de las agencias distribuidoras y de los clientes. Por otra parte, ambas aplicaciones comparten la información de los catálogos de estados, las claves del cliente, los teléfonos y los tipos de teléfono.

De aquí puede concluirse que los dos tipos de información están íntimamente relacionados. Ésta conclusión es lógica si se parte del hecho de que *un cliente de vehículos nuevos será potencial o realmente, un cliente de servicio*, al solicitar mantenimientos de rutina, mantenimientos de garantía, reparaciones de taller u operaciones de hojalatería y pintura.

Al compartir seis tablas los departamentos de ventas y servicio, técnicamente también es idóneo colocar todas estas tablas en la misma base de datos, debido a que si esta información se trata en el mismo espacio de trabajo físico, o "workspace", su manejo será más sencillo y más veloz.

¿Crecerá con el tiempo el tamaño físico de la base de datos? Echando un vistazo a los tipos de información, se observa que tanto las entidades de ventas, catalogos de ventas, entidades de servicio y catálogos de servicio crecen de forma moderada –al estar sujetos a altas, bajas y cambios específicos en el mes-. En donde se centra el estrés de la operación de la base de datos es en todos los movimientos históricos –Ventas, Inventario, Transferencias y Órdenes-, puesto que, por poner un cálculo simple basado en un ejemplo real, si se tienen 150 distribuidoras para la república mexicana de una marca automotriz, y cada marca automotriz tiene como objetivos de ventas 40 órdenes de reparación y 50 unidades nuevas en venta de menudeo, se estarán generando mensualmente 13,500 registros (resultado de multiplicar 150 por 90).

¿Qué ocurriría si esta información tuviera que guardarse anualmente[330]? Entonces se estarían insertando en la base de datos 162,000 registros cada doce meses (resultado de multiplicar 13,500 por 12) lo que ya es una cantidad de registros considerable, algo pesada para manejar en la memoria de una computadora promedio, y para este caso, tomaría una medida un poco más radical, como por ejemplo tener tres espacios de trabajo (uno para la información de ventas, otro para las tablas compartidas y otro para la información de servicio). Afortunadamente, la información histórica de Ventas, Inventario,

[330] ca.

Transferencias y Órdenes expira cada dos meses como máximo extremo –de nada le sirve al corporativo un reporte de ventas ni una órden de reparación del mes anterior-, por lo que la base de datos se depura de información inservible de forma muy periódica y no se necesita un requerimiento tan extremo como este en los servicios de base de datos.

Una vez comprendido porqué estos dos tipos de información deben residir en la misma base de datos, haré referencia al **Anexo 1**, en donde muestro con todo detalle los nombres de las tablas (sean entidades, históricas o catálogos), sus relaciones (sean 1 a 1 o 1 a n), sus llaves primarias de uno o dos campos (marcadas con pk del anglicismo "primary key"), sus llaves foráneas de uno o dos campos (marcadas con fk del anglicismo "foreign key"), sus atributos no llave, la simbología utilizada y su órden de creación, marcado con un número encerrado en un cuadro, que será explicado mas ampliamente un poco más adelante. Tanto las llaves primarias como las llaves foráneas estan enumeradas.

De estas relaciones, ninguna de ellas fue una relación n a m, es decir, no se encontraron relaciones muchos a muchos, lo que hubiera generado un proceso de normalización digno de mencionarse[331]. Desde el principio la base de datos se apegó al conjunto de concepto de "normalidad" en las bases de datos relacionales, concepto que ya es pan y tortilla del programador de nuestros tiempos.

Un dato notable es que en el **Anexo 12** también aparece un diagrama entidad relación. ¿Qué caso tiene presentar en esta tesis dos diagramas de entidad relación? ¿Hay uno "bueno" y uno "malo", o dicho mas formalmente, uno válido y uno inválido? Si el lector analiza y compara con detenimiento el **Anexo 1** y el **Anexo 12**, notará que éste último es el diagrama entidad-relación *para el Cliente*. En los anexos se incluyeron 2 diagramas entidad-relación: uno para la aplicación que residirá en el servidor y otro para el cliente que residirá en la agencia distribuidora. La diferencia entre ambos diagramas reside en que el diagrama del cliente carece de la tabla Agencias, así como las llaves foráneas relacionadas con la tabla Agencias –el atributo agenciaid, o identificadora de la agencia-.

Dicho mas concretamente, una agencia distribuidora no requiere la tabla Agencias, porque por definición, en la distribuidora existe una base de datos pequeña o personalizada, quizas en formato MDB de MSAccess, debido a que maneja información de una razon social. Si es que en la agencia se maneja mas de una razón social, entonces puede darse el caso de que el capturista maneja tantos archivos MDB como sean necesarios. Por tanto, este diagrama entidad-relación del **Anexo 12** *es una estructura sugerida para los proveedores de software certificado*. Como herramienta docente se ha proporcionado un archivo MDB en el "application toolkit" del presente trabajo de investigación, llamado **`WSCLIENT.MDB`** que se apega estrictamente a las definiciones del **Anexo 12**. A final de cuentas, el proveedor certificado es quien decide no solo sus herramientas de desarrollo, o sus bases de datos, o sus métodos de programación a aplicar en la agencia distribuidora, sino también su forma de estratificar y organizar los datos.

En la **sección 4.1** se tiene un bloque etiquetado como "MySQL Database Services". ¿Por qué no se etiquetó este bloque sencillamente como "Base de Datos MySQL"? Esto es porque el mismo motor de MySQL *puede dar alojamiento a múltiples bases de datos a través de un puerto*. El puerto que se abre en la LAN interna es regularmente el 3306, puerto que está disponible para todas las direcciones IP de la subred que no estén aisladas por algún firewall. Entonces, tanto para enviarle comandos al motor de base de datos, como para obtener consultas enteras de una, dos o mas tablas, se utiliza una línea de comunicaciones a través de dicho puerto.

¿Qué utilizamos para generar la base de datos? Para dicha tarea usamos un "script" o programa, que al ejecutarlo, tiene la capacidad de borrar la base de datos anterior si es que existe, y generar unas nuevas tablas, índices, relaciones y catálogos. Este script se encuentra en el **Anexo 2**. El lenguaje que utilizé fue el SQL, con instrucciones de DDL (Data Definition Language). Manejé los comandos DROP DATABASE, CREATE DATABASE, USE DATABASE, DROP TABLE, CREATE TABLE[332], LOCK TABLES, INSERT INTO y UNLOCK TABLES.

[331]Siempre que se encuentra una relación *m a n*, es menester romperla en dos relaciones *1 a n*, a la par que se define una tabla mediadora entre esas dos relaciones. Este paso es parte del proceso de normalización.
[332]Todas las tablas son del tipo `InnoDB`, porque solo sobre este tipo de tablas pueden actuar las reglas de integridad referencial. En MySQL existen diversos tipos de tablas, como `MyISAM`. Para más información puede consultarse directamente la documentación de MySQL en `http://www.mysql.com`

El script del **Anexo 2** lo nombramos **`produccion.sql`**, y tiene una íntima relación con el diagrama Entidad Relación del **Anexo 1**. En el diagrama entidad relación definí unos números encerrados en un cuadro. Pues bien, todas las tablas que tengan el mismo número se dice que están en el mismo nivel de creación. ¿Cuántos niveles de creación tengo? Véase la siguiente tabla:

Nivel de creación	Tabla
1	tblEstados tblTipoDeOperaciones tblTipoDeOrdenes tblTipoDeServicios tblTipoDeVenta tblTipoDePago tblTipoDeTelefonos tblClaveCliente
2	tblAgencias
3	tblVendedores tblInventario tblClientes tblAsesores
4	tblTransferencias tblVentas tblOrdenes tblTelefonos

Tabla 25 Dependencias funcionales entre las tablas de la base de datos

¿Qué significa el hecho de que una tabla esté en tal o cual nivel de creación? **El nivel de creación define la dependencia funcional de la tabla con respecto a otra**. *Por definición, en el nivel de creación 1 todas las tablas son independientes, no dependen funcionalmente de nadie. Las tablas del nivel 2 dependen de las tablas del nivel 1, las tablas del nivel 3 dependen de las tablas del nivel 2 y las tablas del nivel 4 dependen de las tablas del nivel 3*.

Por citar un ejemplo, no puedo definir la tabla tblClientes, la cual esta en el nivel 3 antes de definir la tabla tblClaveCliente, porque el intérprete de MySQL marcará error al momento de hacer una referencia con la sentencia:

```
FOREIGN KEY (clave) REFERENCES tblClaveCliente(cliente) IN UPDATE CASCADE
```

Marcará error debido a que, antes de generar una llave foránea, deben existir previamente los catálogos que le darán ortogonalidad a los datos en los campos referenciados.

Lo que si es válido es definir la tabla tblTelefonos antes de la tabla tblTransferencias, por ejemplo, debido a que ambas no dependen funcionalmente una de otra. Mientras ambas tablas estén en el mismo nivel de creación, podrá conmutarse su órden.

Un dato curioso sobre las tablas que definí como de nivel 1 es que todas ellas son catálogos del sistema. El catálogo tblEstados no requiere mayor explicación, debido a que se trata de los nombres de los estados de la república mexicana; el catálogo tblTipoDeOperaciones almacena los típos de servicio que puede necesitar una unidad en taller: mantenimiento, reparaciones mayores o bien hojalatería y pintura.

El catálogo tblTipoDeOrdenes define el trato monetario que se le dará a la órden de reparación, ya que una órden público siempre se cobra porque es la orden que se levanta en una unidad de un cliente x, la orden interno es cuando un vehiculo recibe mantenimiento antes de ponerse a la venta como vehículo seminuevo, y en este caso el distribuidor no se puede cobrar a sí mismo por ese trabajo aunque si puede reportar dicha orden, y una órden de reparación de garantía es cuando la unidad necesita reparación debido a algún desperfecto de fábrica, por lo que no se le cobrará al cliente el trabajo efectuado sobre dicha unidad.

La tabla tblTipoDeServicios almacena dos valores posibles: paquete y express. Tal como en un restaurante puede solicitarse la comida del día o la comida a la carta, en el taller de servicio puede solicitarse un paquete de diversas operaciones de mantenimiento a un precio especial (afinación, balanceo, bujias, ajuste de motor, ajuste de frenos, transmisión, suspensión, etc) o bien solo una de ellas, dependiendo de la necesidad y el presupuesto del cliente.

Hablando acerca de las ventas de vehiculos nuevos, se tienen diversos tipos de venta en el catálogo tblTipoDeVenta: Ventas a menudeo o al cliente final, ventas a empleados de la agencia con pagos de contado, ventas de menudeo especiales denominadas "satisfactores de compromisos gerenciales" y ventas de menudeo de acuerdo a campañas de ventas o planes corporativos. Por otra parte, se tiene el arrendamiento individual a través de arrendadora independiente. Finalmente, la agencia es capaz de facturar flotillas a nombre personal, a nombre de flotilleros comerciales como refresqueras e industrias de alimentos, ventas a gobierno estatal e incluso, ventas de mayoreo a gobierno federal. Cada unidad vendida debe pertenecer solamente a una de estas categorías en el sistema corporativo

En las ventas de unidades nuevas, solo puede haber tres tipos de pago: pago de contado, pago por financiamiento o pago por arrendamiento. Esta información está guardada en la tabla tblTipoDePago

Hablando acerca de los teléfonos, es un error de normalización querer asociar solo uno o dos telefonos a un cliente, porque habrá clientes que solamente proporcionen un telefono de casa, pero habrá clientes que podrán proporcionar telefonos de casa y oficina, así como teléfonos celulares del conductor o chofer responsable de esa unidad. Esa es la razón por la que se creó el catálogo tblTipoDeTelefonos, para regular esa política y aceptar solamente un conjunto de tipos de teléfono determinados.

Finalmente, en el último catálogo de la capa 1 se encuentra la tabla tblClaveCliente. Debe especificarse si el cliente es una persona física (señor, señora o señorita), una persona moral, o bien si el cliente es un representante en el extrajero, si es una venta de negocios.

¿Qué llaves primarias existen en la base de datos? La tabla *tblAgencias* tienen el identificador **agenciaid**. Los *vendedores* tienen el identificador **rfcVendedor+agenciaid** debido a que un vendedor puede vender unidades de una, dos o más razones sociales. En el *inventario* se identifica la unidad con **serieVehiculo**. En la tabla de *tblTipoDeVenta* la llave primaria tiene el nombre **tipoDeVenta**. En la tabla *tblTipoDePago* la llave primaria tiene el nombre **tipoDePago**. La tabla *tblEstados* tiene la llave primaria **estado**. La tabla *tblClientes*, por su parte, tiene la llave compuesta **rfc_o_curp_cliente+agenciaid**, debido a que un mismo cliente pudo haber comprado su unidad en una agencia y recibir servicio en otra, o bien un cliente puede comprar dos unidades en dos agencias diferentes. La tabla *tblVentas* tiene el identificador único **serieVehiculo.** La tabla *tblClaveCliente* tiene la llave **clave**. La tabla *tblOrdenes* tiene la llave primaria compuesta **ordenid+agenciaid** debido a que un mismo número de folio de orden puede existir en una, dos o mas agencias, sin restricción. La tabla *tblTipoDeTelefonos* tiene la llave primaria **tipodetelefono**. La tabla *tblAsesores* tiene la llave compuesta **rfcAsesor+agenciaid**, debido a que un asesor de servicio puede trabajar para dos razones sociales diferentes, dentro de una misma localidad física. La tabla *tblTipoDeOperaciones* tiene la llave **tipodeoperacion**. La tabla *tblTipoDeOrdenes* tiene la llae primaria **tipodeorden**, y la tabla *tblTipoDeServicios* tiene la llave primaria **tipodeservicio**.

¿Qué relación existe entre la tabla de clientes y la de teléfonos? Estrictamente, un cliente puede tener 0, 1 o más teléfonos, por lo que el programa de Web Services validará que existan 1, 2 o más teléfonos.

En el "script" producción.sql se encuentran también definidos los tipos de datos propuestos para cada uno de los atributos en las entidades, catálogos e históricos. Ahora bien, ¿cómo se ejecuta el script en la base de datos? En el **Anexo 6**, "Guía de Instalación", se tiene un procedimiento para ello. En los pasos 8 y 9 se muestra que se debe copiar el programa producción.sql en el directorio de trabajo de MySQL, para después ejecutarlo en el cliente de MySQL. A continuación se da una reproducción de estos pasos.

8	Copiar la base de datos en mysql	Copiar **`C:\jwsdp-1.3\webapps\produccion\tesis\produccion.sql`** En **`C:\mysql\bin`**

9	Instalar la base de datos en mysql	Ir a ventana Command Posicionarse en **C:\mysql\bin** Ejecutar **C:\mysql\bin>mysql --user=root < produccion.sql**

Una vez definida la base de datos, e instalada en el servidor de MySQL, podrían preguntarse asuntos técnicos como por ejemplo *¿de que forma se pueden explotar los servicios y recursos de MySQL, si el lenguaje de programación seleccionado fue Java 2 Standard Edition de Sun Mycrosystems*? Sun provee una tecnología para manejar este tipo de servicios de una forma directa y sencilla, la tecnología Java DataBase Connectivity (JDBC), bien conocida por la comunidad de Analistas de Sistemas y desarrolladores de productos de software. JDBC fija un método de siete pasos basicos, que nos garantizan un exitoso uso de cualquier servicio de base de datos. Ese método consta de[333]:

1. **Cargar el controlador JDBC**. El controlador de MySQL para Java 2 Standard Edition está disponible de forma gratuita en la página de MySQL (http://www.mysql.com/)
2. **Definir el URL de conexión**. Tal como un cartero necesita saber no solo la calle y el número exterior, sino también el número interior de un domicilio, un cliente de JDBC necesita saber no solo la dirección IP y el puerto en donde se encuentra el servicio atendiendo solicitudes, sino tambien el nombre del espacio de trabajo, el usuario y el password que definen el perfil del usuario en la base de datos.
3. **Establecer la conexión**.
4. **Crear un objeto Statement de las bibliotecas de Java 2 Standard Edition**.
5. **Ejecutar la consulta o la actualización**.
6. **Procesar los datos**.
7. **Cerrar la conexión**.

Este procedimiento es óptimo cuando solamente ingresa un usuario a la vez en la base de datos. ¿Qué ocurre cuando, **concurrentemente, varios usuarios usan el sistema al mismo tiempo**? Para este caso es indispensable un programa que administre las conexiones, ¿Por qué? La razón principal es que abrir una conexión a una base de datos es un proceso retardado. En pequeñas consultas, podría haber mayor retraso en abrir la conexión que al realizar la consulta. Por consecuencia, tiene sentido volver a utilizar los objetos `Connection` en las aplicaciones que se conectan repetidamente a la misma base de datos.

Es por eso que se necesita una clase de Java para asociar las conexiones, es decir, asignar previamente conexiones a bases de datos, y reciclarlas conforme los clientes se conecten. Los servlets y las páginas JSP pueden beneficiarse significativamente de esta clase dado que la base de datos a la que se conecta cualquier página JSP o servlet típicamente se conoce de antemano, (es decir, se especifica en el método init). Por ejemplo, el conjunto de servlets mostrados en el **Anexo 3** muestra una ganancia siete veces mayor al hacer uso de esta clase para la asociación de conexiones.

Una clase para la asociación de conexiones debe realizar las siguientes tareas:

1. Asignar previamente las conexiones.
2. Administrar las conexiones disponibles.
3. Asignar nuevas conexiones.
4. Aguardar a que una conexión quede disponible.
5. Cerrar las conexiones cuando sea necesario.

El código de la clase ConnectionPool para administración de conexiones del sistema se muestra en el listado de los programas del **Anexo 3**. Estas clases las obtuve de http://www.pearsonedlatino.com/ServletsJSP, y las modifiqué para que se adaptaran al JDBC específico de MySQL.

¿Qué tuve que hacer para que la versátil clase ConnectionPool trabajara en mi beneficio? Si cualquier programador descarga esta clase de la página supracitada, encontrará que dicha clase define la utilización de drivers de Oracle y Sybase solamente. El cambio que realizé es que al método "Constructor", o el método que genera objetos a partir de los planos de la clase, le

[333] Vid. http://java.sun.com/products/jdbc/drivers.html

incluí la información del driver, el connection string, el parámetro de conexiones iniciales así como el parámetro de máximas conexiones. Dentro de esa misma clase ConnectionPool modifiqué el método "`makeNewConnection`", quien se encarga tanto de cargar el driver de MySQL como de establecer la conexión en red para transferencia de comandos y datos. Es importante destacar en este punto que la mayoría de los fabricantes cargan los drivers de la base de datos con la instrucción

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

O bien, con la instrucción:

```
Class.forName("com.sybase.jdbc.SybDriver");
```

En cambio, el método varía un poco para MySQL, y la línea equivalente es:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Finalmente, utilizé el método para establecer la conexión. Su forma es bien conocida:

```
DriverManager.getConnection(connString);
```

Gracias a este cambio, la clase ConnectionPool trabaja en armonía con mi Web Service –cuya puerta de entrada es **`ServicioHTTP.java`**-, quien en el método init genera un objeto ConnectionPool, el cual se pasa como parámetro en los métodos, y cada método obtiene una conexión de base de datos MySQL a partir de él. Dicho cambio, en realidad trivial, no alteró en forma alguna la metodología ni la funcionalidad de la clase `ConnectionPool`. La forma en que trabaja dicha clase –que pertenece al paquete de clases `coreservlets` - es[334]:

1. **Asignar previamente las conexiones.** Esta tarea se ejecuta en el método constructor de la clase. El establecer más conexiones previamente agiliza las cosas si habrá muchas peticiones concurrentes, pero provoca un retardo inicial. Como resultado, un servlet que asigne previamente muchas conexiones deberá generar la asociación de conexiones desde su método `init`, y debería asegurarse de que el servlet haya establecido sus valores de inicio antes de que se lleve a cabo una petición "real" de un cliente. El siguiente código utiliza vectores para almacenar conexiones inactivas disponibles, y conexiones ocupadas (que no estén disponibles). Daremos por hecho que makeNewConnection utiliza un URL, un nombre de usuario y una contraseña almacenados con antelación, y simplemente ejecutaremos al método `getConnection` de `DriverManager`.
   ```
   conexionesDisponibles=new Vector (conexionesIniciales);
   conexionesOcupadas=new Vector();
   for (int i=0; i< conexionesIniciales; i++) {
         conexionesDisponibles.addElement(makeNewConnection());
   }
   ```
2. **Administrar las conexiones disponibles**. Si una conexión es solicitada y hay alguna disponible, se debe colocar en la lista de conexiones ocupadas y devolverse como resultado del método o función. La lista de conexiones ocupadas se utiliza para verificar los límites de la cantidad total de conexiones, así como cuando se instruye al conjunto de asociaciones para cerrar de forma explícita todas las conexiones. Cabe indicar que las conexiones pueden fracasar, por lo que antes de devolver una conexión se debe confirmar si esta se encuentra abierta. En caso de que no sea el caso, es prudente descartar la conexión y repetir el proceso. Descartar una conexión abre un lugar que puede ser utilizado por procesos que necesiten una conexión cuando el límite de conexiones se haya alcanzado, por lo que deberá utilizarse el elemento notifyAll para indicarle a todos los subprocesos en espera que vean si pueden proceder (por ejemplo, a crear una nueva conexión).
   ```
   public synchronized Connection getConnection()
      throws SQLException {
      if (!conexionesDisponibles.isEmpty()) {
      Connection conexionExistente =
      (Connection)conexionesDisponibles.lastElement();
      int ultimoIndice = conexionesDisponibles.size() - 1;
      conexionesDisponibles.removeElementAt(ultimoIndice);
      if (conexionExistente.isClosed()) {
         notifyAll();
   ```

[334] HALL, Marty. Servlets y Java Server Pages: Guía práctica, pp. 501-503

```
            return(getConnection()); // Repetir el proceso
        } else {
            conexionesOcupadas.addElement(conexionExistente);
            return(conexionExistente);
        }
    }
```

3. **Asignar nuevas conexiones**. Si se necesita una conexión, no hay una disponible y no se ha llegado al límite de conexiones, entonces se debe iniciar un subproceso de segundo plano para asignar una nueva conexión. Luego, se debe esperar a que se genere la primera conexión disponible, ya sea o no la recién asignada.

```
if ((conexionesEnTotal() < maxConexiones) && !conexionPendiente) {
   generarConexionEnSegundoPlano();
}
try {
      wait();
} catch (InterruptedException ie) {}
return (getConnection());
```

4. **Aguardar a que una conexión quede disponible**. Esta situación sucede cuando no hay una conexión disponible y se ha alcanzado el límite de conexiones. Esta espera debe realizarse sin una asignación contínua. La metodología natural es utilizar el método `wait,` que da un bloqueo en la sincronización y suspende al subproceso hasta un `notify` o `notifyAll.` Dado que notifyAll podría provendir de diversas fuentes, los subprocesos que reinicien su ejecución necesitarán verificar si pueden proceder. En tal caso, la forma más sencilla de realizar esto es repetir recursivamente el proceso de intentar obtener una conexión.

```
try {
      wait();
} catch (InterruptedException ie) {}
return (getConnection());
```

   Podría ser que no se desee dejar en espera a los clientes y, en lugar de ello, **arrojar una excepción** cuando no haya una conexión disponible y que se haya alcanzado el límite de conexiones. En tal caso, se puede hacer lo siguiente:

```
throw new SQLException ("Se ha alcanzado el límite de conexiones");
```

5. **Cerrar las conexiones cuando sea necesario**. Cabe destacar que las conexiones se cierran cuando se les hace la recolección de basura del java virtual machine, por lo que no siempre tendrán que cerrarse explícitamente. Pero en ocasiones se querrá tener un mayor control en este proceso.

La clase **ConnectionPool** quedará copiada en el directorio

`C:\jwsdp-1.3\webapps\produccion\WEB-INF\classes\coreservlets`

En resumen, los pasos que se siguieron para **proveer los servicios de base de datos** en ésta sección, fueron:

- Diseñar la base de datos del **Anexo 1**.
- Escribir las instrucciones de SQL con instrucciones DDL.
- Crear un nuevo espacio de trabajo en MySQL y crear las tablas con el programa del paso II.
- Instalar los manejadores de JDBC en el sistema operativo.

Ésta base de datos provee la cimentación de la infraestructura del servicio. Las siguientes secciones mostrarán cómo se diseño y programó dicha interfaz para actuar sobre la base de datos, de manera que se diera respuesta a la problemática expuesta en el **Capítulo 1**.

4.3 Creación de la interfaz para efectuar transacciones en el sistema.

En la **sección 4.2** y el **Anexo 1** se ha definido a detalle la base de datos en la que se recibirán los reportes de ventas de los distribuidores, la información de inventario y transferencias, así como las órdenes de reparación y la información relacionada con los clientes, las agencias, los vendedores de unidades nuevas y los asesores de servicio. Ésta base de datos podrá ser consultada directamente por el corporativo, sin intermediarios, usando quizás un sencillo manejador como el Microsoft ODBC para MySQL –qué a su vez se conecta con aplicaciones del Microsoft Office 2003, o bien el cliente gráfico de MySQL, para construir desde simples archivos ASCII para enviarlos por FTP, hasta formas y reportes en Microsoft Access, hojas de cálculo en Microsoft Excel, Mail Merge en Microsoft Outlook, o sendmail de Linux.

Si la base de datos de produccion instalada en los servicios de base de datos de MySQL puede explotarse a través de diversas herramientas con conectividad directa vía TCP/IP, ¿**Por qué no asignarle al distribuidor una cuenta en la base de datos y ahorrarse la interfaz Web services**? ¿Por qué no propusimos una arquitectura cliente servidor como la siguiente?

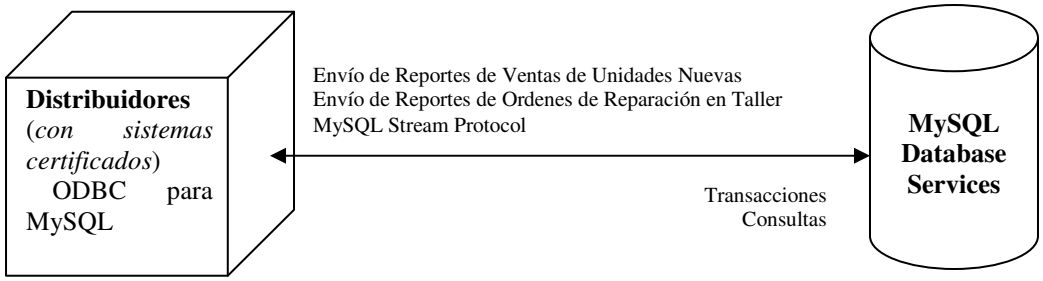


Ilustración 46 Arquitectura inoperable Cliente/Servidor de base de datos

Existen muchas razones, pero las de más peso son las siguientes:

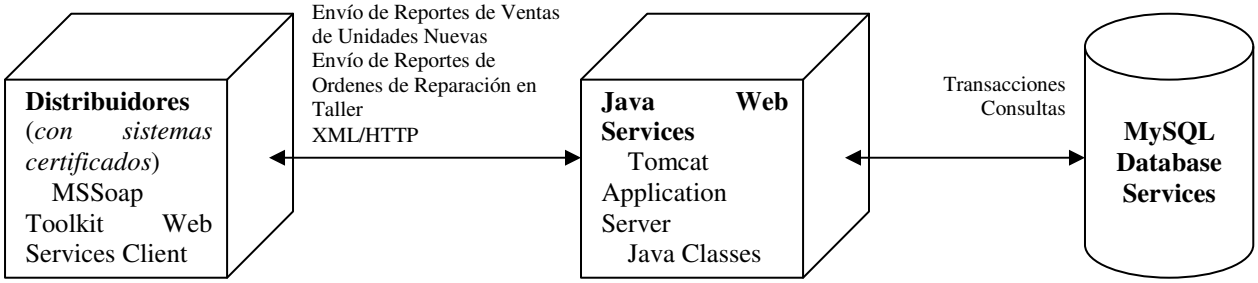
1. Los MySQL database services son provistos en el **puerto 3306** del servidor donde reside, y tomando en cuenta que la **red corporativa** está protegida no con uno sino con **varios firewalls** –nacionales y norteamericanos- en la arquitectura de TCP/IP (tanto de Hardware como de Software) sería prácticamente imposible abrir esa regla tan solo para una aplicación de desarrollo nacional.
2. La **red del distribuidor** también está protegida con uno o varios firewalls, tanto de Hardware, como de Software, y no todos los distribuidores estan dispuestos a abrir un puerto, y menos en estos días que abundan los ataques de virus y hackers en puertos abiertos o desatendidos.
3. Todos los **distribuidores tendrían que aprender el lenguaje SQL**, las técnicas para abrir la base de datos vía ODBC y tendrían que hacer programación muy elaborada en sus sistemas certificados tan solo para enviar reportes de ventas y órdenes de reparación, lo que les generaría una carga económica adicional. No todas las agencias distribuidoras cuentan con un departamento de sistemas, ni tienen un programador que responda a los cambios aún más ligeros de programación.
4. Los distribuidores no tendrían el tiempo necesario para **reaccionar ante un cambio en la base de datos**, suponiendo que el corporativo decidiera desechar el motor de MySQL, y adquirir un motor de base de datos comercial como Microsoft SQL Server, Oracle 9i, Sybase, u otro[335].
5. Siendo MySQL la excepción, todos los usuarios de servidores de base de datos tienen que pagar conforme al número de conexiones en el día desde diversos puntos. El corporativo de la industria automotriz no tendría forma de administrar solamente **50 licencias, por ejemplo, y administrarlas entre 150 usuarios**, sino que tendría que pagar por cada distribuidor que se conecte, aunque lo haga esporádicamente. El número de distribuidores tiende a

[335]Este fenómeno es clásico en los corporativos de la industria automotriz. Cuando un sistema catalogado como "prototipo" ha dado prueba de su eficacia, utilidad, versatilidad y concordancia con la meta de la compañía –vender unidades, vender refacciones, vender servicios, vender franquicias, hacer clientes, hacer dinero-, se procede a migrarlo a servidores más veloces, con motores de bases de datos más poderosos. En éste caso ¡La compañía gira aprobaciones para la compra de todo tipo de licencias de harware/software y no escatima en gastos!

aumentar, no a disminuir, y el pagar una conexión para nuevos usuarios es una concesión que la industria automotriz evita al querer administrar los recursos disponibles.

6. Bajo el esquema propuesto anteriormente, la base de datos estaría sujeta a intensos ataques desde los nodos de los distribuidores –la información de los clientes, las ventas y servicio es estrictamente confidencial, y no puede quedar a la deriva de esta manera-. Desde luego, no se desconfía de los empleados de la distribuidora, mas si de los proveedores de la distribuidora que en ocasiones no son bienintencionados y buscan información para vendérsela a la competencia.
7. El distribuidor no sabría que hacer exactamente en caso de que la base de datos le reportara un error. Es decir, no sabría directamente si el error fue provocado por la ausencia de un dato, por un dato mal escrito o por datos en un órden erróneo.
8. En vista de que no todos los distribuidores soportan sus operaciones en los sistemas operativos de Microsoft -hay distribuidoras que tienen puras terminales de hp ux, linux o apple-, tendrían que adquirir una PC tan solo para instalar el ODBC de MySQL para enviar sus reportes de ventas y órdenes de reparación, cosa que es absurda si ya se cuentan con recursos disponibles.
9. Regularmente, la conectividad de ODBC en nodos remotos (con baja velocidad de transferencia) tiende a perderse, lo que obligaría a que el distribuidor tenga que estar reestableciendo la conexión continuamente. Tal caso resultaría en reducción en el desempeño del software de los sistemas certificados.
10. Nadie garantiza que la base de datos MySQL sea un estándar el día de mañana. Nadie sabe si dentro de unos años más este motor deje de recibir soporte técnico, o si su uso sea de un costo superior a otras marcas debido a su privatización.

Por tales razones, el corporativo necesito de una solución que abatiera todos estos problemas. La piedra que mató no 2 pájaros de un tiro, sino diez, son los web services. Por eso el esquema propuesto quedó como sigue:



Ésta solución consta de las siguientes características:

1. Los XML Web services son provistos en el puerto 80 del servidor donde reside. Por tanto, a pesar que la red corporativa está protegida no con uno sino con varios firewalls –nacionales y norteamericanos- en la arquitectura de TCP/IP (tanto de Hardware como de Software) el puerto 80 está abierto para páginas web y aplicaciones tanto de desarrollo nacional como de desarrollo norteamericano.
2. A pesar que el distribuidor también está protegiendo su red con uno o varios firewalls, tanto de Hardware, como de Software, esto no representa ningun problema al momento de establecer una conexión de HTTP en el lado del cliente. De hecho, eventualmente hasta podría instalarse un XML Web service en el lado de la distribuidora, y *ningún concesionario tendría base para sentirse inseguro con respecto a proveer una conexión por el puerto 80*.
3. *Ningún distribuidor tiene que aprender el lenguaje SQL*, ni las técnicas para abrir la base de datos vía ODBC. Los proveedores de sistemas certificados tienen que hacer la tarea de comunicarse con el XML Web services a través del protocolo SOAP vía HTTP, lo que es menor en costos y complejidad a tener que aprender el lenguaje SQL tan solo para mandar reportes de ventas y órdenes de reparación.
4. Si el día de mañana el corporativo decidiera desechar el motor de MySQL, y adquirir un motor de base de datos comercial como Microsoft SQL Server, Oracle 9i, Sybase, u otro, este cambio sería completamente transparente para el distribuidor, debido a que la conexión la seguirían haciendo los Java Web Services, y la interfaz de XML SOAP seguiría trabajando exactamente igual.
5. El corporativo de la industria automotriz tiene forma de comprar un número limitado de licencias, 50 por ejemplo, para administrarlas entre 150 usuarios, sin tener que pagar por nuevos distribuidores o por aquellos que se conecten esporádicamente debido a que reportan sus ventas por segundos o terceros canales. El corporativo adquirirá nuevas

licencias de conexión solo si el tiempo de espera para que una conexión quede libre tiende a incrementarse con el transcurso del tiempo. De hecho, el corporativo podría instalar nuevos servidores de bases de datos para balancear las cargas de trabajo, y esto también permanecería transparente para el distribuidor.

6. El Web service puede protegerse usando varias capas de seguridad. Por definición, se tienen dos capas de seguridad en el sistema –una en el nivel de la base de datos y otra en las reglas de seguridad de los routers y firewalls-, aunque se pueden añadir nuevas capas, como encriptar la línea de datos del enlace de HTTP –tecnología conocida como Secure Sockets Layer o SSL[336]-, crear nuevos certificados a través de proveedores de seguridad en la infraestructura de internet, y así sucesivamente. La información de los clientes, las ventas y servicio es estrictamente confidencial y se pueden seguir añadiendo tantas capas de seguridad como el corporativo crea necesario. En este requisito no hay límites.
7. El distribuidor sabe perfectamente qué hacer al momento de que se genera un error o excepción en el sistema. El distribuidor, sabría qué transacción no se procesó correctamente, y qué fue lo que ocasionó el error, por ejemplo, la ausencia de un dato, un dato mal escrito o datos en un órden erróneo.
8. Si los distribuidores soportan sus operaciones en sistemas operativos ajenos a Microsoft –sea unix, linux o apple-, no tienen porqué adquirir una PC con WindowsXP tan solo para instalrle el Microsoft SOAP Toolkit para enviar sus reportes de ventas y órdenes de reparación. Bastará con que consigan la biblioteca de SOAP de su lenguaje de programación y sistema operativo correspondiente para construir el cliente. Por ejemplo, Java provee la tecnología multiplataforma JAXM (Java XML Messaging) y JAXP (Java XML Processing), con lo que la distribuidora podrá reutilizar sus recursos conforme crea necesario y conforme a las sugerencias del proveedor de sistemas certificado.
9. Las conexiones de HTTP son sin estado, lo que les permite abrirse y cerrarse contínuamente, por lo que los XML Web Services de Java resultan la solución ideal en una red de baja velocidad de transferencia, como lo son la mayoría de las redes corporativas de las firmas de giro automotriz.
10. Si la base de datos MySQL deja de ser un estándar el día de mañana, siempre habrá otras opciones de base de datos –algunos apuestan que Posgress será la base de datos estándar dentro de unos años-. No importa qué base de datos sea impuesta el día de mañana, siempre y cuando el fabricante provea *el driver de JDBC pertinente*.

[336]El desarrollar un sofisticado sistema de seguridad no es el objetivo primario del presente trabajo de investigación. Para mas información sobre este tema deben consultarse tesis, libros y manuales que traten con todo detalle los algoritmos de encriptación, metodologías de seguridad y políticas de protección en las redes de computadoras modernas.

En la hipótesis propuesta se observa que los Java Web services reciben los XML de Ventas de Unidades Nuevas, y de Órdenes de Reparación vía HTTP antes de hacer las transacciones y consultas a la base de datos. Aquí surgen las cuestiones, ¿Qué tipos de XML se reciben? ¿Qué tipos de XML se envían? ¿Qué información está contenida en los XML? ¿Qué formato tienen? Para dar respuesta a las primeras dos preguntas, se debe establecer que **éste sistema se basará en la arquitectura RPC que da respuesta a cada solicitud de la siguiente forma:**

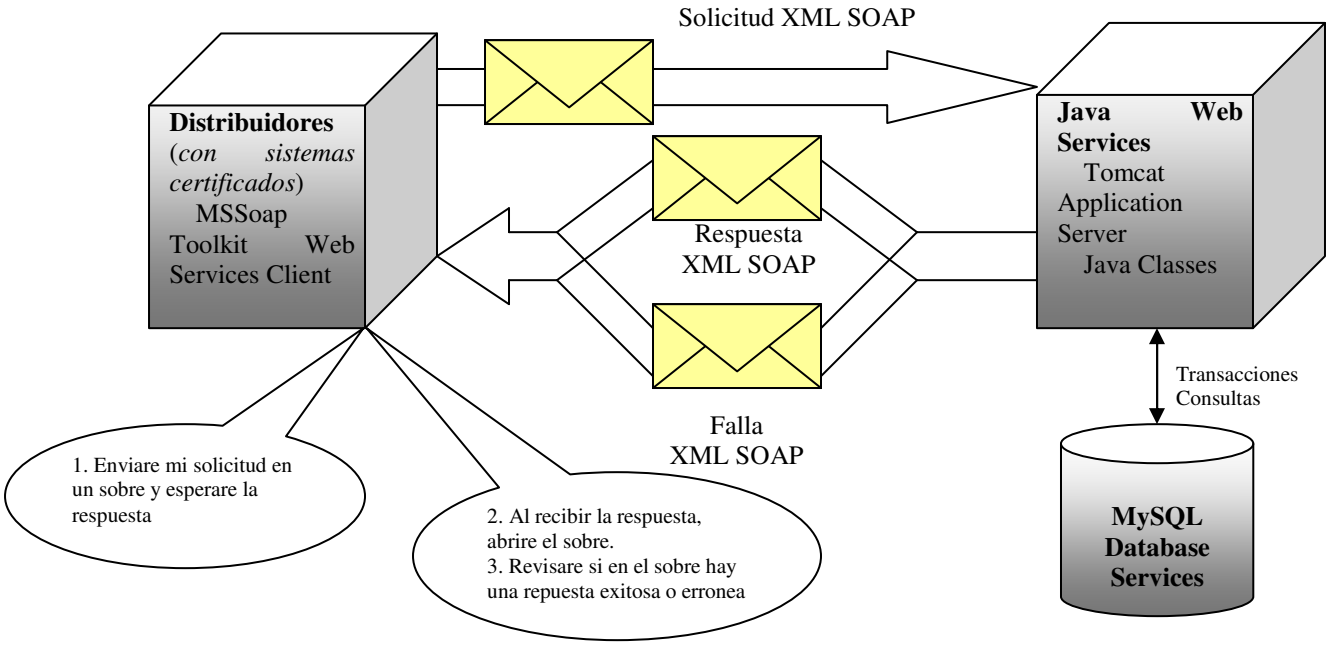


Ilustración 47 Los Web Services proveen una arquitectura RPC basada en mensajes

Del diagrama anterior se comprende que a cada solicitud de SOAP, o solicitud de XML, corresponde una respuesta, también de SOAP o respuesta de XML. Además, se tiene que existen dos tipos de respuestas SOAP: la respuesta que se genera cuando la transacción es exitosa y cuando no es el caso.

¿Qué tipos de transacciones existen? Al estudiar el diagrama entidad-relación de la base de datos del **Anexo 1**, *hicimos el siguiente diseño de transacciones para el área de ventas y el área de servicio:*

Transacción de Ventas	**Operación en la Base de Datos**
AltaUnidad	Genera un insert en *tblInventario*
BajaUnidad	Genera un delete en *tblInventario*
AltaInventario	Genera uno o más insert en *tblInventario*
AltaVendedor	Genera un insert en *tblVendedores*
BajaVendedor	Genera un delete en *tblVendedores*
ListaVendedores	Genera un select en *tblVendedores*
AltaCliente	Genera un insert en *tblClientes* y uno o más insert en *tblTelefonos*
BajaCliente	Genera un delete en *tblClientes* y uno o más delete en *tblTelefonos*
ListaClientes	Genera un select en *tblClientes* y otro en *tblTelefonos*
ReportaVenta	Genera un insert en *tblVentas*
CancelaVenta	Genera un delete en *tblVentas*
Transferencia	Genera un update en *tblInventario*, y un insert en *tblTransferencias*

Tabla 26 Métodos transaccionales de Ventas

Transacción de Servicio	Operación en la Base de Datos
AltaCliente	Genera un insert en *tblClientes* y uno o más insert en *tblTelefonos*
BajaCliente	Genera un delete en *tblClientes* y uno o más delete en *tblTelefonos*
ListaClientes	Genera un select en *tblClientes* y otro en *tblTelefonos*
AltaAsesor	Genera un insert en *tblAsesores*
BajaAsesor	Genera un delete en *tblAsesores*
ListaAsesores	Genera un select en *tblAsesores*
AltaOrden	Genera un insert en *tblOrdenes*
ImportesOrden	Genera un update en *tblOrdenes* sobre ingresos y utilidades
CierraOrden	Genera un update en *tblOrdenes* sobre cierre
CancelaOrden	Genera un delete en *tblOrdenes*

Tabla 27 Métodos transaccionales de servicio

Desde luego, estas diecinueve transacciones no se encuentran en un entorno ideal en el que no existen peligros de seguridad, por lo que se exige la información del usuario y del password en cada transacción. Además, los *select*, *insert*, *update* y *delete* **son los productos finales que entrega el servicio**, por lo que **antes de ejecutarlos se verifican reglas de validación al menos en tres niveles***, a saber, la estructura y tipos de datos de xml dadas por el corporativo, las reglas de negocios dadas por el corporativo, y las reglas de integridad referencial dadas por la base de datos.*

¿Cuál es el parámetro principal de cada transacción? *Dicho parámetro es el encabezado* **SOAPAction**, quien aparece en el encabezado del método **POST** transferido por **HTTP**. Para nuestra solución, utilizamos **SOAPAction** para que TOMCAT identificara a cual metodo de Java Web services se esta enviando una solicitud de información o solicitud de transacción. Ese importante encabezado es leído en el Servlet **ServicioHTTP.java**[337] con las instrucciones:

```
String soapActionString = request.getHeader("soapaction");
soapActionString=soapActionString.substring(1,soapActionString.length()-1);
//osw.write("soapaction\n"+soapActionString+"\n");

// Get all the headers from the HTTP request
       MimeHeaders headers = getHeaders(request);

// Get the body of the HTTP request
InputStream is = request.getInputStream();

// Preparar la respuesta para todos lo metodos
response.setHeader("Content-Type", "text/xml; charset=utf-8");
javax.servlet.ServletOutputStream sos = response.getOutputStream();
SOAPMessage reply = null;

try
{
       if (soapActionString.equals("http://www.unam.mx/AltaUnidad")){
              AltaUnidad objeto = new AltaUnidad(headers,is);
              objeto.ejecuta(connectionPool);
              reply = objeto.respuesta();
       } else if (soapActionString.equals("http://www.unam.mx/BajaUnidad")) {
              BajaUnidad objeto = new BajaUnidad(headers,is);
              objeto.ejecuta(connectionPool);
              reply = objeto.respuesta();
       } else if (soapActionString.equals("http://www.unam.mx/AltaInventario")) {
              AltaInventario objeto = new AltaInventario(headers,is);
              objeto.ejecuta(connectionPool);
              reply = objeto.respuesta();
```

En segundo plano, ¿Qué parámetros necesita cada una de las diecinueve transacciones para ejecutarse? En el **Anexo 4** esbozo el número de parámetros, su órden de aparición, y también su órden jerárquico. La idea de representarlos como un árbol está muy ligada al concepto de Document Object Model (DOM), el cual establece la metodología de construir objetos

[337] El código completo de éste programa se encuentra en el **Anexo 5** del presente trabajo de investigación.

de tipo jerárquico a partir de documentos XML. De hecho, para procesar un XML, es más facil almacenarlo en un árbol de tipo DOM y consultar la información de sus nodos que simplemente analizar el XML con recursos de programación estructurada como "ciclos for", o "ciclos while".

Dentro del **Anexo 4**, se tienen tres árboles, o tres DOM para cada transacción. De esta manera se define una estructura de la solicitud que el Java Web service espera, y dos estructuras que el cliente debe esperar a cambio.

Analizando los árboles del **Anexo 4**, se observa que *existen dos tipos de árboles: los de número de nodos fijos y los de número de nodos variables*. Los árboles con un número de nodos fijos son los SOAPRequest de AltaUnidad, BajaUnidad, AltaVendedor, BajaVendedor, BajaCliente, ReportaVenta, CancelaVenta, AltaAsesor, BajaAsesor, AltaOrden, ImportesOrden, CierraOrden, ListaClientes, Transferencia, ListaAsesores, ListaVendedores y CancelaOrden. Los SOAPResponse que también tienen un número fijo de nodos son AltaUnidadRespuesta, BajaUnidadRespuesta, AltaInventarioRespuesta, AltaVendedorRespuesta, BajaVendedorRespuesta, AltaClienteRespuesta, BajaClienteRespuesta, ReportaVentaRespuesta, CancelaVentaRespuesta, AltaAsesorRespuesta, BajaAsesorRespuesta, AltaOrdenRespuesta, ImportesOrdenRespuesta, CierraOrdenRespuesta, TransferenciaRespuesta y CancelaOrdenRespuesta. Además, todos los SOAPResponse de tipo Fault también tienen un número de nodos fijo, y son del mismo tipo para los diecinueve métodos transaccionales de web services.

Por otra parte, no todos los árboles de los mensajes SOAP tienen un número de nodos fijos. También son necesarios los árboles con un número variable de nodos. En este caso está el método AltaInventario. En su nodo inventario pueden aparecer 1, 2 o más elementos llamados serieVehiculo, debido a que este método funciona para una distribuidora que tiene 30 unidades en inventario de la misma forma que una distribuidora que tiene 180 (qué ya es una cantidad un poco exagerada pero posible). Otro caso de árboles variables está en el SOAPRequest para el AltaCliente, en donde su nodo teléfonos puede tener uno, dos o más elementos tipo telefono. A pesar que los elementos teléfono deben tener solo tres elementos cada uno (tipodetelefono, lada, y numero), puede existir un número ilimitado de teléfonos. Esto satisface la regla de negocios que establece que entre más telefonos se tienen de un cliente, existe mejor probabilidad de levantarle la encuesta para evaluación de servicio. El SOAPResponse de ListaClientesRespuesta es un caso similar al supracitado.

El SOAPResponse de ListaAsesoresRespuesta también tiene nodos variables, porque si bien es cierto que un asesor tiene los nodos hijos fijos de rfcAsesor y nombre, pueden existir 0, 1, 2 o más asesores trabajando para una razon social. El SOAPResponse de ListaVendedoresRespuesta está en un caso idéntico debido a que sus nodos fijos son rfcVendedor y nombre, y pueden existir 0, 1, 2 o más vendedores en el departamento de vehiculos nuevos en la distribuidora.

De este análisis de nodos se puede decir que los **métodos ListaAsesores, ListaClientes y ListaVendedores cumplen una función especial, al ser informativos para la agencia distribuidora**, para saber con qué entidades cuenta en su base de datos. *De hecho, sin información acerca de estas entidades no es posible reportar ventas ni órdenes de reparación*. En un determinado momento puede perderse la sincronía de datos (es decir, que el cliente pierda la información de Clientes, Asesores y Vendedores, y requiera recuperarla desde los web services) y para esto es que han sido definidos estos métodos transaccionales. Este caso no aplica para el inventario, puesto que si el distribuidor llega a perder irreversiblemente los datos de las unidades de forma local, puede recuperarlo directamente de los archivos de unidades facturadas que se le envía diariamente a través de otro sistema, o bien puede recuperarlo desde la factura impresa que siempre se encuentra en la guantera de la unidad.

Los métodos se deben usar individualmente y en el órden que se requiera, pero como sugerencia, antes de usar el método ReportaVenta, debe haberse utilizado previamente el método AltaUnidad o AltaInventario para dar de alta la unidad en la tabla tblInventario, AltaCliente para tener la información válida de un cliente, con uno, dos o más teléfonos, y AltaVendedor para tener la información correspondiente en la tabla tblTeléfonos. Por su parte, antes de usar el método AltaOrden, se debe usar previamente el método AltaCliente y AltaAsesor. Además después de usar el método AltaOrden, se podrá usar el método ImportesOrden, CierraOrden y CancelaOrden sobre la orden dada de alta. Finalmente, se puede decir que antes de usar el método Transferencia debe haberse utilizado el método AltaUnidad y si la unidad ya fue reportada como vendida por alguien, entonces ese alguien debe usar el método CancelaVenta.

El **Anexo 4** muestra una representación gráfica de los objetos Java de tipo DOM Tree (árboles del modelo de objeto para documentos). La transformación de un documento XML en un DOM Tree es conocido como unmarshall o

"descomposición". El proceso inverso, de un DOM Tree hacia un documento XML es conocido como marshall o composición[338].

A partir de los DOM Tree que se tienen en el **Anexo 4**, ¿cómo se verían el DOM Tree, de AltaAsesor, por ejemplo, si se le aplicara un proceso de marshall? En el **Anexo 7** se tienen documentos de instancia de esos DOM Tree, pero para mayor claridad, a continuación se mostrará la comparación entre ese DOM Tree y su correspondiente documento XML.

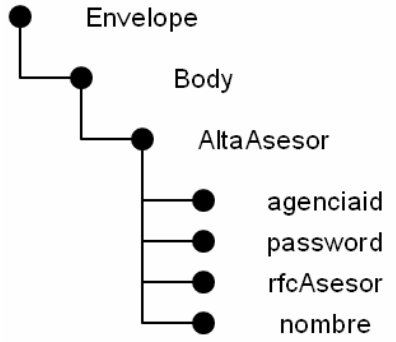
DOM Tree	XML
	<?xml version="1.0" encoding="UTF-8" standalone="no"?><SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema" xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body><AltaAsesor><agenciaid>M1188</agenciaid><password>moreno</password><rfcAsesor>MAHE150980</rfcAsesor><nombre>Erika Marchan HernÃ¡ndez</nombre></AltaAsesor></SOAP-ENV:Body></SOAP-ENV:Envelope>

Tabla 28 XML DOM Tree VS Documento XML

El DOM Tree y el documento XML anteriormente mostrados manejan exactamente la misma información y guardan exactamente la misma estructura, pero el DOM Tree es obviamente más legible que el XML puro tanto para un programa como para una persona.

Si éste tipo de XML SOAP Request es observado desde un navegador de páginas web, gana legibilidad y se tendría que las siguientes entidades son lógicamente equivalentes:

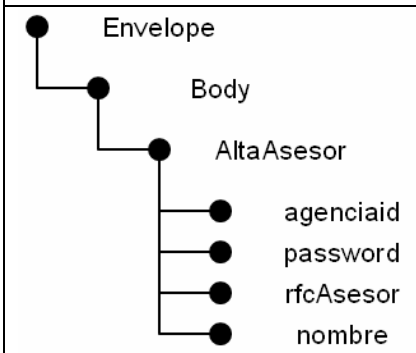
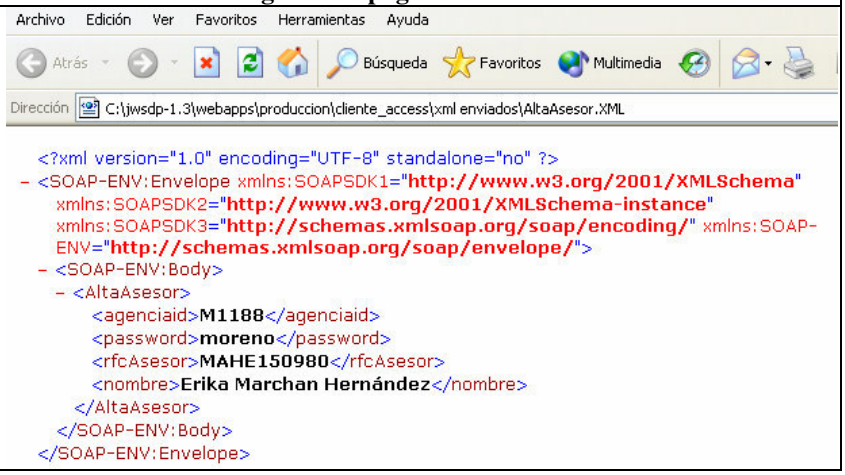
DOM Tree	Vista de XML en navegador de páginas web
	

Tabla 29 XML DOM Tree VS Documento XML en Navegador

En el ejemplo anterior se puede ver que los nodos agenciaid, password, rfcAsesor y nombre adquieren valores y son nodos simples. El nodo AltaAsesor es un nodo complejo, porque tiene cuatro elementos hijos. Otra propiedad observable en los documentos XML es que **los nodos Envelope y Body pertenecen al namespace oficial definido por la W3C para los XML Schemas, mientras que los nodos AltaAsesor, agenciaid, password, rfcAsesor y nombre no cuentan con namespace alguno, para facilitar la programación de la interfaz proxy**, *aunque no habría ningún inconveniente en definirles su propio namespace si los estándares de la industria automotriz así lo requirieran*.

[338] ARMSTRONG, Erick, et al., The Java Web Services Tutorial, p. 16

A continuación se mostrarán otros DOM Trees de **Anexo 4** comparados con los XML del **Anexo 7**:

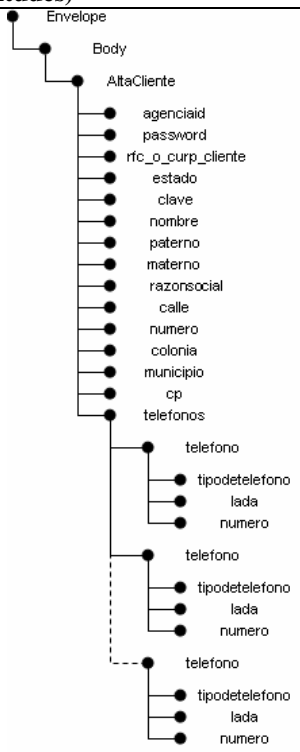
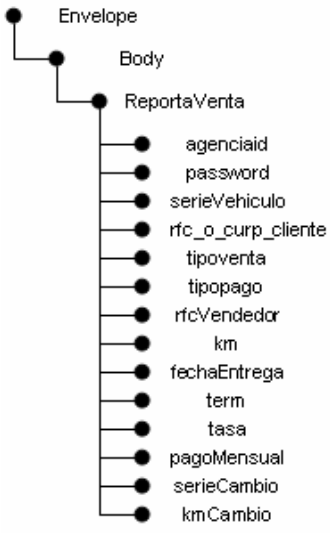
DOM Tree de SOAP Requests (Solicitudes)	**Vista de XML en navegador de páginas web**
	<?xml version="1.0" encoding="UTF-8" standalone="no" ?> - <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema" xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> - <SOAP-ENV:Body> - <AltaCliente> <agenciaid>M1188</agenciaid> <password>moreno</password> <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente> <estado>DF</estado> <clave>sr</clave> <nombre>Samuel</nombre> <paterno>Camacho</paterno> <materno>Garcia</materno> <razonsocial>Samuel Camacho García</razonsocial> <calle>Insurgentes Sur</calle> <numero>1325</numero> <colonia>San Angel</colonia> <municipio>Magdalena Contreras</municipio> <cp>01010</cp> - <telefonos> + <telefono> - <telefono> <tipodetelefono>compradorcel</tipodetelefono> <lada>04455</lada> <numero>52202612</numero> </telefono> + <telefono> </telefonos> </AltaCliente> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
	<?xml version="1.0" encoding="UTF-8" standalone="no" ?> - <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema" xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> - <SOAP-ENV:Body> - <ReportaVenta> <agenciaid>M1188</agenciaid> <password>moreno</password> <serieVehiculo>3LS1M423ASL895941</serieVehiculo> <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente> <tipoventa>vmenudeo</tipoventa> <tipopago>financiamiento</tipopago> <rfcVendedor>CAAA120859</rfcVendedor> <km>200</km> <fechaEntrega>2004-07-20T12:45:24</fechaEntrega> <term>12</term> <tasa>3.5</tasa> <pagoMensual>3500</pagoMensual> <serieCambio>1KP1M423AML895941</serieCambio> <kmCambio>120000</kmCambio> </ReportaVenta> </SOAP-ENV:Body> </SOAP-ENV:Envelope>

Tabla 30 XML DOM Trees VS otros documentos XML

En estos XML DOM Trees se observa también que al aplicar el marshall, se generan los elementos Envelope y Body en el namespace de los elementos de SOAP, y tanto el nodo ReportaVenta como sus nodos hijos carecen de namespace o

calificativo. Otra cosa que se observa es que el navegador de Web provee de controles marcados con un mas "+" y "-" para extender y comprimir los nodos como sea necesario, y esta simple característica permite observar de una forma cómoda la estructura y los datos del documento XML.

Los tres ejemplos anteriores –AltaAsesor, AltaCliente y ReportaVenta- son muestras de cómo están formadas las solicitudes de SOAP. Solamente al usar un navegador para visualizar los documentos XML se puede afirmar que dichos son fácilmente legibles para un ser humano. Para completar el cuadro comparativo, a continuación se muestran ejemplos de las respuestas de SOAP.

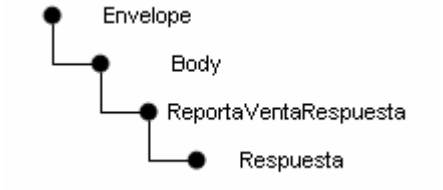
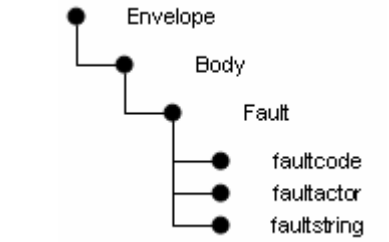
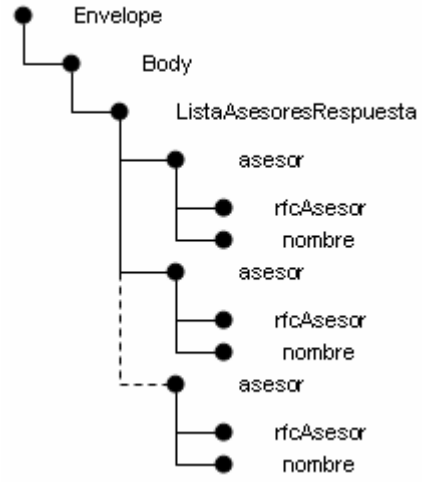
DOM Tree de SOAP Responses (Respuestas)	**Vista de XML en navegador de páginas web**
	<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> <SOAP-ENV:Body> <m:ReportaVentaRespuesta xmlns:m="http://www.unam.mx/ReportaVentaRespuesta"> <Respuesta>9999-ReportaVenta es Exitoso</Respuesta> </m:ReportaVentaRespuesta> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
	<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> <SOAP-ENV:Body> <SOAP-ENV:Fault> <faultcode>SOAP-ENV:Server</faultcode> <faultactor>http://www.unam.mx/ReportaVenta</faultactor> <faultstring>5140 - El vehiculo 3LS1M423ASL895941 no ha sido dado de alta en el inventario. Utilice el metodo AltaUnidad o AltaInventario</faultstring> </SOAP-ENV:Fault> </SOAP-ENV:Body> </SOAP-ENV:Envelope>
	<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> <SOAP-ENV:Body> <m:ListaAsesoresRespuesta xmlns:m="http://www.unam.mx/ListaAsesoresRespuesta"> <asesor> <rfcAsesor>MAHE150980</rfcAsesor> <nombre>Erika Marchan Hernandez</nombre> </asesor> <asesor> <rfcAsesor>RAIR760930</rfcAsesor> <nombre>Raul Ramirez Izquierdo</nombre> </asesor> </m:ListaAsesoresRespuesta> </SOAP-ENV:Body> </SOAP-ENV:Envelope>

Tabla 31 XML DOM Trees de respuesta y su representación XML en Navegador

Los últimos tres ejemplos anteriores muestran los **tres diferentes tipos de respuesta en formato XML SOAP**. El antepenúltimo caso -ReportaVentaRespuesta- se trata de una confirmación simple del método, por ejemplo, de una venta. Este tipo de mensajes son los más comunes. Por otra parte, si se trata de pasar por alto una regla en la validación del xml, en las reglas de negocios o en la integridad referencial de la base de datos, se generará el mismo tipo de mensaje de error en los diecinueve métodos, conocido como SOAP Fault response, ilustrado en el penultimo caso.

En el último caso se tiene un listado del catálogo obtenido con el método ListaAsesoresRespuesta. Una respuesta similar se encuentra al recibir los mensajes ListaVendedoresRespuesta y ListaClientesRespuesta, resultado de aplicar los métodos ListaVendedores y ListaClientes, respectivamente. Si se desea seguir haciendo éstas comparaciones visualmente, entonces debe contrastarse el **Anexo 4** contra el **Anexo 7** del presente trabajo de investigación.

En conclusión, tanto las solicitudes como las respuestas están codificadas en el lenguaje XML, en la gramática particular SOAP.

Al haberse definido el papel de los documentos XML en el intercambio de información entre los Web services y los clientes, así como el papel de los MySQL Database services en la arquitectura desarrollada, procederemos a **descomponer la solución en Java Web Services para analizar cada uno de sus elementos por separado**. Nótese el siguiente diagrama:

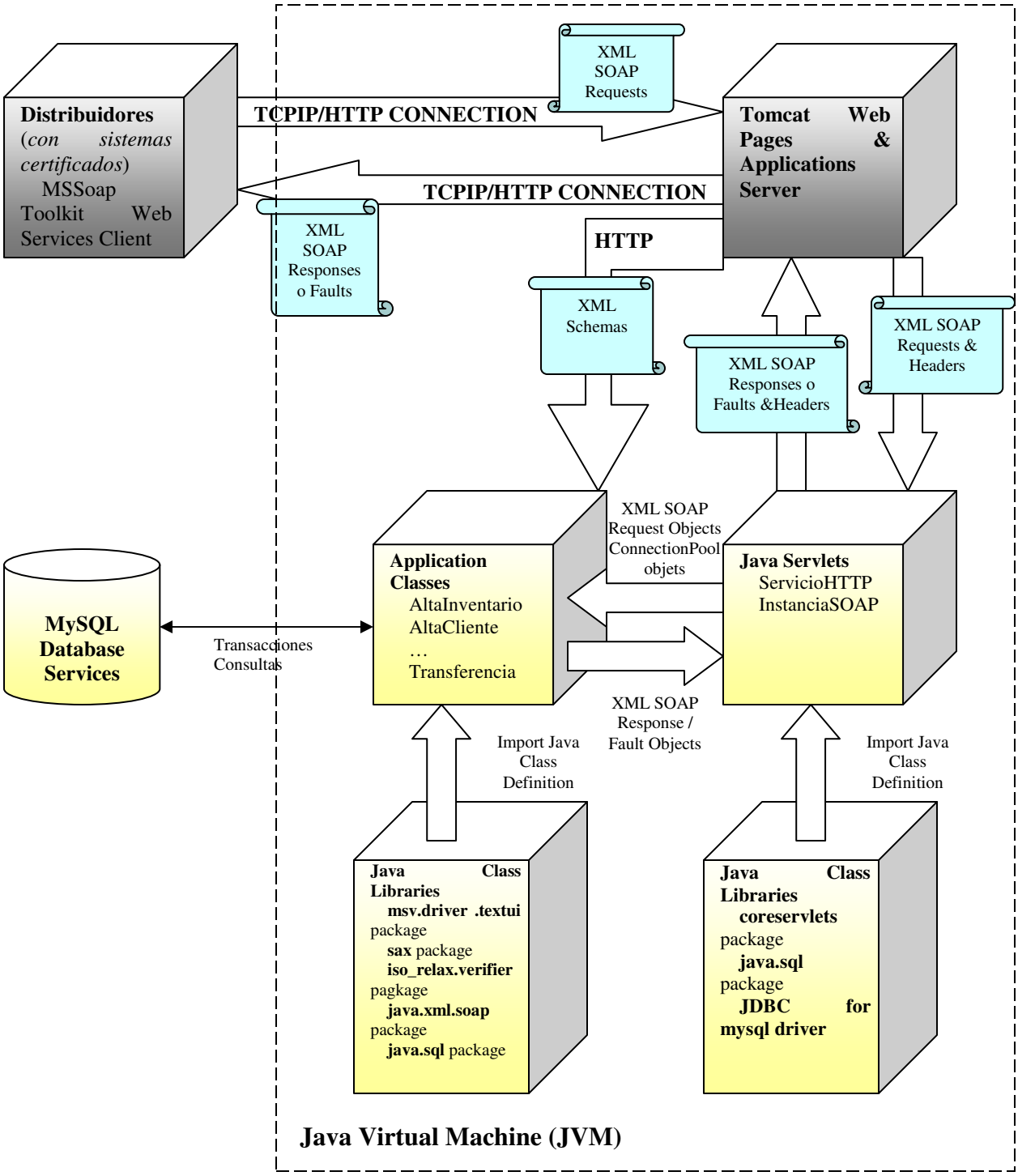


Ilustración 48 Descomposición de la solución en Java Web Services para su análisis

En el diagrama anterior se representa el ciclo que guarda la transacción RPC. **Aquí se observa que el cliente en el distribuidor se comunica con una Java Virtual Machine sin que siquiera lo note, debido a que dicha JVM provee un servicio HTTP en el protocolo de red TCP/IP**.

El ciclo está compuesto de los siguientes pasos.

1. Los clientes envían solicitudes de XML SOAP al Tomcat Web Pages & Applications Server.
2. Tomcat envía los XML SOAP Requests a los Servlets alojados en el directorio de aplicación, por ejemplo a ServicioHTTP alojado en el directorio de aplicación producción.
3. ServicioHTTP.java elabora un objeto ConnectionPool y un objeto XML SOAP Request. Además, obtiene el nombre del método de Web services que viene en el encabezado.
4. ServicioHTTP envía al método correspondiente el objeto ConnectionPool y el objeto XML SOAP Request.
5. Las clases de aplicación reciben los XML SOAP Requests así como el objeto ConnectionPool. Validan los XML SOAP Requests con el XML Schema correspondiente.
6. Las clases de aplicación, descomponen el objeto XML SOAP Request. Validan los parámetros y las reglas de integridad referencial. Después hacen las transacciones o consultas a la base de datos usando el objeto ConnectionPool.
7. Si la transacción es exitosa, la clase de aplicación elabora un SOAP Response exitoso. En caso contrario, elabora un SOAP Fault.
8. Envía el SOAP Fault de regreso al servlet ServicioHTTP.java.
9. El servlet ServicioHTTP le pone un encabezado de HTTP y lo regresa al Tomcat Web Pages & Applications Server.
10. El Tomcat regresa la respuesta al cliente usando el protocolo HTTP como medio de transporte.

Dentro de este diagrama debe notarse que tanto los Java Servlets como las Application Classes utilizan las Bibliotecas de Clase de Java. Dichas Bibliotecas regularmente vienen agrupadas en paquetes o "packages", y se agrupan cuando tienen funciones y metas relacionadas. **Así, existen packages para validación de XML, para marshall y unmarshall de XML, para transacciones de SQL, para drivers de MySQL, para definición de servlets, etc**. En este sentido, programar las clases de aplicación no requieren "volver a inventar la rueda". Bastará con usar las bibliotecas de clases preconstruidas para tratamiento de XML, manejo de SQL., o cualquier otra tarea que se necesite.

Las clases de aplicación están impresas en el **Anexo 5**. ¿Cómo están construidas? Se pueden hacer diagramas de clase para las clases de aplicación. Por ejemplo, la clase de aplicación AltaInventario tiene la siguiente apariencia:

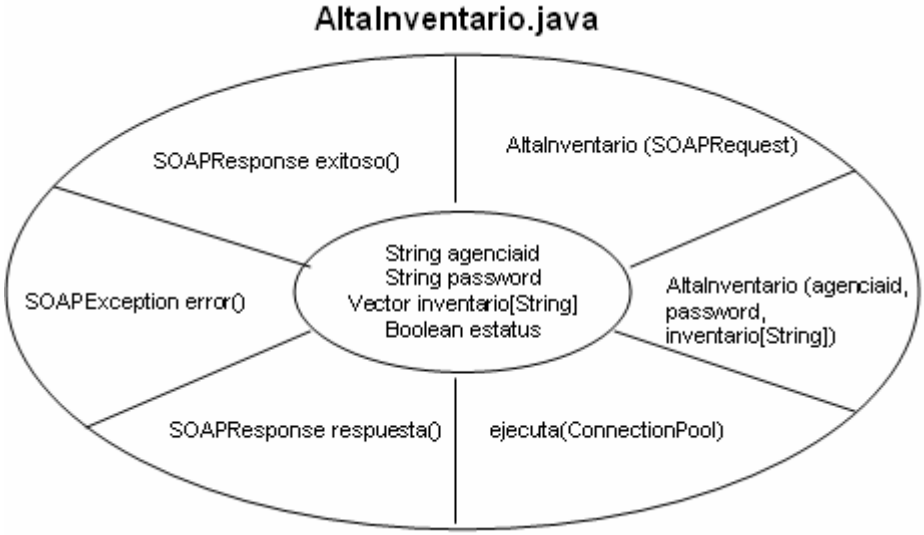


Ilustración 49 AltaInventario.java

En esta clase se observa que sus variables de clase son agencia, password, estatus, y un arreglo de cadenas llamado inventario. El método constructor le asigna los valores a las variables de clase a partir del SOAP Request o a partir de valores constantes, ya que soporta sobrecarga en los métodos. El método ejecuta usa el objeto ConnectionPool y las variables de clase para realizar operaciones sobre la base de datos. El método ejecuta también actúa sobre la variable de

clase estatus, la pone en "TRUE" cuando la transacción ha sido exitosa o "FALSE" en caso contrario. Finalmente, el método respuesta ejecutará el método exitoso() si el estatus es true, y error() si el estatus es false(). Esta medida devolverá el tipo de mensaje SOAP Response correspondiente.

Análogamente, se tienen otros diagramas de clases de aplicación. Tal es el caso del AltaUnidad.java. Este caso es casi idéntico a **AltaInventario.java**, pero la diferencia radical reside en que en vez te tenerse un arreglo de strings llamado inventario, se recurre a una objeto String para manejar información de solamente una unidad.

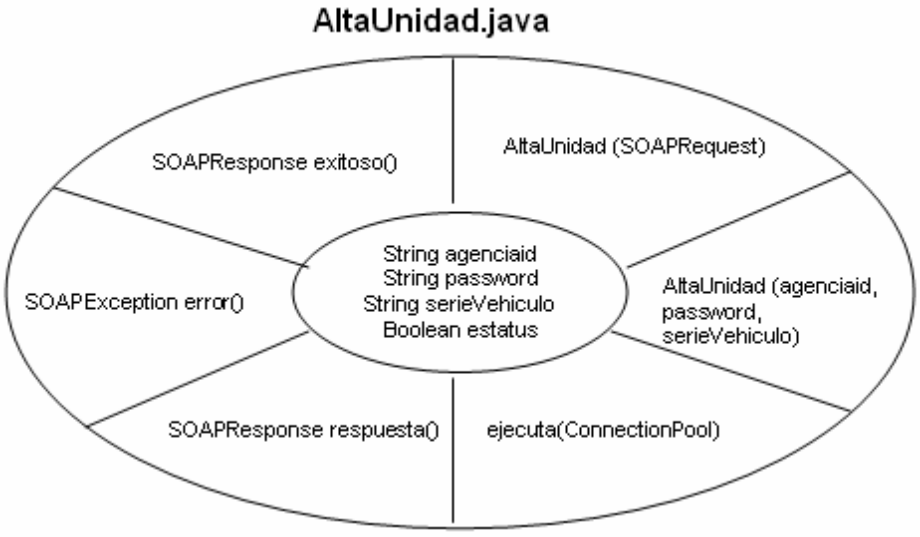


Ilustración 50 AltaUnidad.java

Se tiene un tercer ejemplo del diagrama de clases de aplicación. Este ejemplo corresponde a la clase AltaCliente.java. Aquí, se observa que una de sus variables de clase es un objeto de tipo Vector, llamado teléfonos. Éste vector no almacena objetos String, como la clase AltaInventario, sino que almacena objetos de tipo telefono:

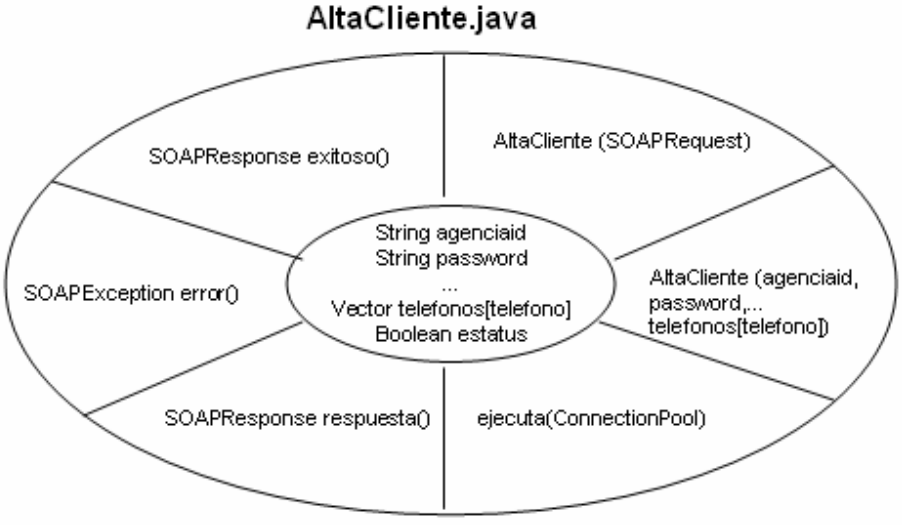


Ilustración 51 AltaCliente.java

Si alguien busca información respecto a la clase telefono en la documentación de Java 2 Standard Edition, seguramente no la encontará porque este es un objeto que definí para almacenar mis atributos personalizados en el objeto teléfono: lada, número y tipodetelefono. Adicionalmente diré que esta clase no tiene métodos significativos:

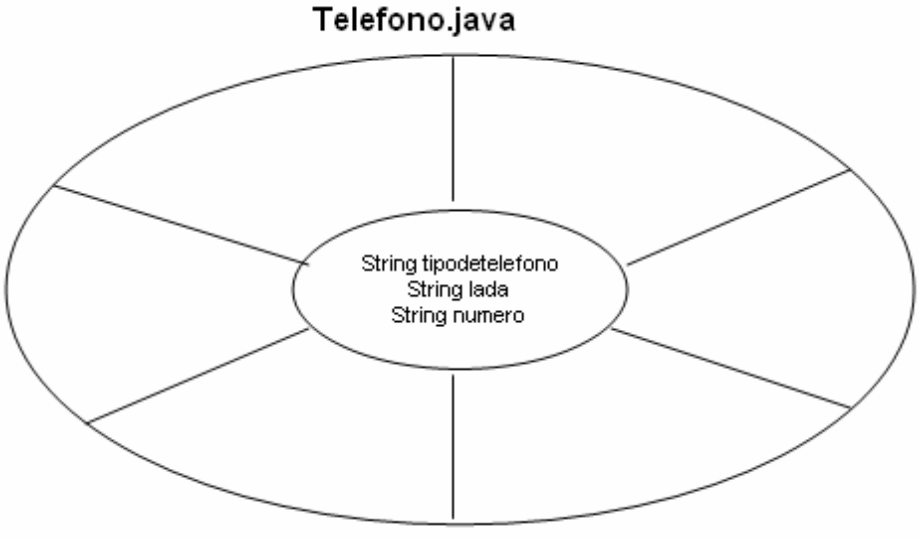


Ilustración 52 Telefono.java

Las demás clases de aplicación siguen la misma metodología. De hecho hasta se podría definir una superclase que las describa a todas ellas.

Éste es un panorama general de la aplicación, y las clases de Java involucradas en el proceso de atención de transacciones o métodos de Web services. La siguiente sección comentará cómo valida los datos el sistema a partir de los XML Schemas.

4.4 Diseño de reglas de validación de datos en XML Schemas

En la interfaz Java Web services para la industria automotriz se tienen tres tipos de validación:

1. Validación de estructura y datos de los XML SOAP Requests que recibe el Tomcat Server.
2. Validación de reglas de negocios.
3. Validación de reglas de integridad referencial en la base de datos.

Las reglas de negocios se revisan en las clases de Java y las reglas de integridad referencial se validan en la base de datos. ¿De qué forma se validan la estructura y los datos de los XML SOAP Requests? Para ilustrarlo, se invita a pensar en el caso de llenar una solicitud para obtener una Visa en la Embajada Americana. Allí, el personal consular ofrece solicitudes al personal que previamente ha hecho una cita. Las solicitudes tienen un determinado número de cajas de texto en donde el aspirante debe escribir determinados datos. ¿Qué ocurre si en el campo de "nombre" el aspirante escribe su número de seguro social? ¿Qué ocurre si el aspirante no contesta la pregunta "viaje de placer/viaje de negocios" con alguna de las dos opciones? ¿Qué ocurre si el aspirante no declara un contacto en el extranjero, o el número de días que permanecerá en los Estados Unidos? Pues bien, aún estando en la fila y antes de entrar a la oficina, está un agente que se encarga de revisar las solicitudes de una por una. Si la solicitud está bien contestada, entonces el agente otorga sello y marca en la solicitud y el aspirante podrá entrar a la oficina para entregar su solicitud, y otra documentación. Si la solicitud no está razonablemente llenada, el agente regresa al aspirante a la fila para que haga sus correcciones antes de volver a revisar la solicitud.

Este sencillo ejemplo ilustra muy bien que es la validación de estructura y datos de la solicitud. El XML SOAP Request también es como una solicitud para obtener Visa, solo que en este caso tiene información relacionada con la industria automotriz. **Es indispensable que la información sea válida aún antes de procesarse en el sistema por tratarse de datos críticos para la organización**. Las razones para validar los SOAP Requests en este caso son al menos las siguientes:

- Si no se envían los datos de agenciaid y password en el orden solicitado, se corre el riesgo de que el sistema rechaze la transacción y tenga base para argumentar que el usuario no está registrado en el sistema, o que se está tratando de enviar información ilegal al corporativo.

- Si se envían los datos de agenciaid y password correctamente, pero no se envían los demás datos, entonces se corre el riesgo de que se inserte información nula en el sistema.
- No son opcionales los parámetros en los métodos transaccionales de web services. Todos los parámetros son obligatorios.
- Si se envían datos ficticios, ilegibles o aleatorios, entonces se corre el riesgo de que se inserten datos espúreos en la base relacional, lo que se conoce como "basura". Una base de datos con basura tiende a incrementarse rápidamente en tamaño, y ello ocasionaría invertir tiempo en la depuración y reducción en el tiempo de respuesta.
- Al no enviarse el numero de serie de vehiculo en el formato correcto, entonces no hay forma de reportar la venta a Estados Unidos, debido a que el corporativo siempre revisa que exista la unidad en las bases de datos correspondientes (sea la unidad mexicana, argentina, brasileña, europea o norteamericana).
- Al no enviarse los datos del cliente correctamente, entonces se corre el riesgo de que se deseche la información de ventas y órdenes por no poder comprobar que existe un destinatario final de productos y servicios.
- Al no enviarse en el XML SOAP Request los tipos de datos correspondientes a la base de datos, se tiene la desventaja de que en el servidor de base de datos se generará una transacción errónea. A pesar de que en la base de datos usé un método de recuperación de fallas conocido como begin-commit-rollback, lo que no puedo recuperar es el tiempo muerto de procesamiento de 50 transacciones en un minuto. No hay que meter un clip en el contacto solo para saber si hay corriente.
- Las etiquetas en el XML SOAP Request deben estar perfectamente definidas dentro del namespace de su pertenencia.
- Los conjuntos permitidos de valores para determinadas claves enviadas están definidos dentro de catálogos
- Al no enviarse los datos en el formato solicitado, se corre el riesgo de que el servidor los interprete de manera diferente a como los ve el cliente.

Esto revela porqué se requieren validar los datos de la solicitud en formato XML SOAP. Así, tal como en la embajada americana hay un agente consular en la puerta, *nosotros también colocamos un agente consular en la entrada del servicio Web de Java*. ¿Qué pasos llevamos a cabo para tal efecto? El primer paso es recordar el siguiente detalle en el esquema propuesto de la arquitectura solución:

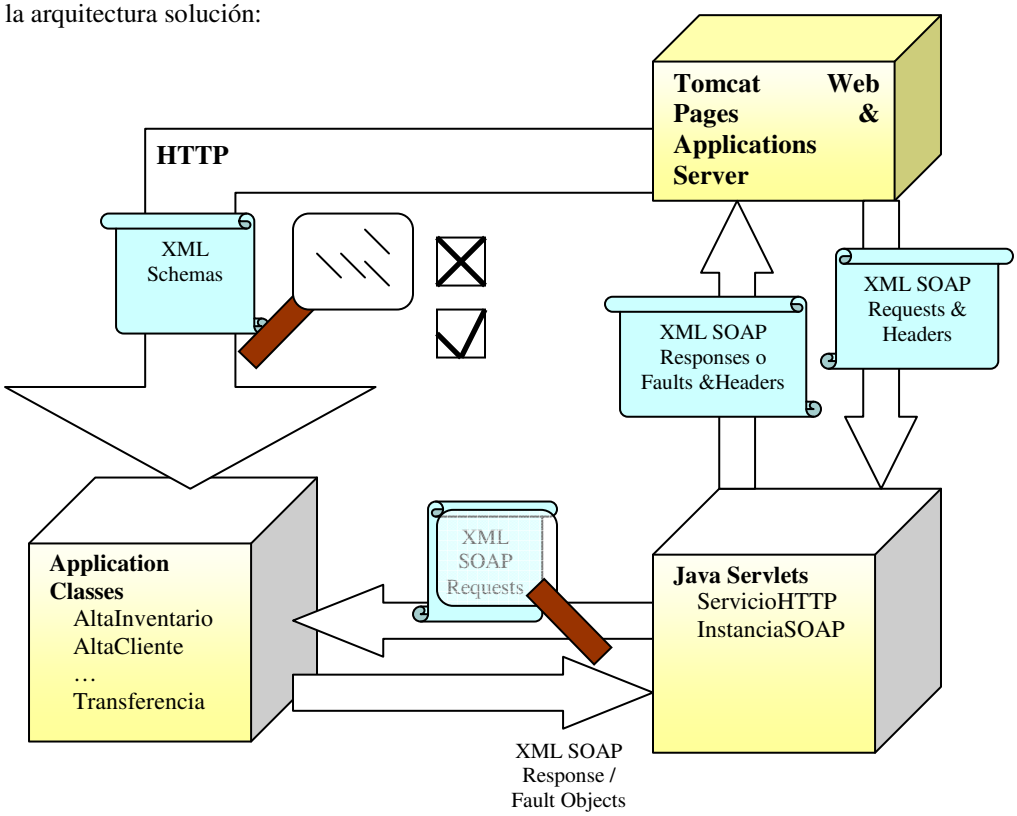


Ilustración 53 Revisión de documentos XML SOAP con XML Schemas

Aquí se observa que las clases de Java AltaInventario, AltaCliente, Transferencia, etc. reciben dos tipos de información: reciben las solicitudes XML SOAP, pero también obtienen XML Schemas del servidor vía HTTP. Se tiene un XML Schema para cada clase, por ejemplo para AltaInventario.java se tiene **AltaInventario.XSD**, para AltaCliente.java se tiene **AltaCliente.XSD**.

¿Qué son esos archivos XSD, exactamente? En la sección 3.2.1 dedicada los XML Schemas, mencioné que un archivo XSD también es un archivo XML que tiene directivas almacenadas para el Agente de validación. Así como el Agente consular tiene un manual de políticas y reglas en su bolsillo –el cual consulta regularmente para recordar las políticas de la embajada o instruir al usuario-, de la misma forma **los XSD son pequeños pero eficientes manuales de políticas y reglas que le recuerdan al usuario de Web services las políticas y reglas de la industria automotriz**.

Ahora bien, estas directivas por sí mismas no servirían de nada sin alguien que las haga valer. Es decir, si no existiera un agente consular en la entrada de la embajada, no todos los usuarios entregarían sus solicitudes de visa correctamente llenadas. En el Java Web service, ¿Quién es el "agente consular"? ¿Quién se encarga de aplicar los XSD Schemas a los XML SOAP Requests? Sun Microsystems provee tres poderosas bibliotecas de clases o "packages" para hacer esta tarea sencilla. Los packages involucrados son los siguientes:

- `org.iso_relax.verifier.*`
- `com.sun.msv.driver.textui.ReportErrorHandler`
- `org.xml.sax.*`

Estas clases, proveen objetos *para saber si un documento de instancia, el XML SOAP Request, en este caso, se adapta al XML Schema*. En caso de que no lo sea, se detecta que tipo de error existe, si hay un error en el XML SOAP Request (programado con SAXException) o si hay un error en el micromanual de políticas (programado con VerifierConfigurationException).

Un detalle interesante es que el **Anexo 5** está titulado como "Sistema Java Web Services Version 02". Si ya se tiene la versión 2 del sistema, ¿Dónde quedó la Versión 1? ¿Porqué se requirió una nueva versión? La versión 1 quedó obsoleta debido a que la versión 2 valida los XML Soap Requests con los XML Schemas. ¿Cómo es esto posible? Pues bien, en cada método constructor después de transformar el XML SOAP Request en un DOM Tree y antes de extraer las variables de clase del DOM Tree, se realiza la validación, con cuatro líneas de código:

```
// Código Agregado para la versión 2.0 del sistema
// para validar datos desde la llegada del xml
VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
Schema schema =
factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaAsesor.xsd");
Verifier verifier = schema.newVerifier();
verifier.verify(part);
// fin de código para version 2.0
```

El código anterior aplica para el método AltaAsesor, y cada XML SOAP Request de AltaAsesor será validado con el XML Schema llamado AltaAsesor.xsd.

Si bien los DOM Trees gráficos propuestos en **Anexo 4** son un excelente material docente para entender la relación entre los XML y las transacciones, ¿Cómo puede describirse formalmente la estructura de ucvn documento XML? En la **sección 3.2.1** se mencionó que los XML Schemas son la mejor opción para esta tarea. De hecho, *un solo XML Schema puede describir a toda una familia de documentos de instancia de XML. Normalmente, los documentos de instancia que se validan son los XML SOAP Requests que recibe el servidor*. Por definición, no es necesario validar los XML SOAP Responses con un XML Schemas, debido a que los web services los generan y por tanto se asume que no contienen errores estructurales. Si hicieramos un cuadro comparativo parcial de los XML SOAP Requests de Instancia contra sus XML Schemas correspondientes se tendría algo como lo siguiente:

XML Schemas	XML SOAP Request de instancia

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaAsesor" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave" form="unqualified" />
                <xs:element name="password" type="tns:styClave" form="unqualified" />
                <xs:element name="rfcAsesor" type="tns:styRFC" form="unqualified" />
                <xs:element name="nombre" type="tns:styNombre" form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-
    instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaAsesor>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfcAsesor>MAHE150980</rfcAsesor>
        <nombre>Erika Marchan Hernández</nombre>
      </AltaAsesor>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

```
- <xs:element name="AltaCliente" form="unqualified">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="agenciaid" type="tns:styClave" form="unqualified" />
        <xs:element name="password" type="tns:styClave" form="unqualified" />
        <xs:element name="rfc_o_curp_cliente" type="xs:string" form="unqualified" />
        <xs:element name="estado" type="xs:string" form="unqualified" />
        <xs:element name="clave" type="xs:string" form="unqualified" />
        <xs:element name="nombre" type="xs:string" form="unqualified" />
        <xs:element name="paterno" type="xs:string" form="unqualified" />
        <xs:element name="materno" type="xs:string" form="unqualified" />
        <xs:element name="razonsocial" type="xs:string" form="unqualified" />
        <xs:element name="calle" type="xs:string" form="unqualified" />
        <xs:element name="numero" type="xs:string" form="unqualified" />
        <xs:element name="colonia" type="xs:string" form="unqualified" />
        <xs:element name="municipio" type="xs:string" form="unqualified" />
        <xs:element name="cp" type="xs:string" form="unqualified" />
      - <xs:element name="telefonos" form="unqualified">
        - <xs:complexType>
          - <xs:all>
            - <xs:element name="telefono" form="unqualified" minOccurs="1" maxOccurs="unbounded">
              - <xs:complexType>
                - <xs:sequence>
                    <xs:element name="tipodetelefono" type="xs:string" form="unqualified" />
                    <xs:element name="lada" type="xs:unsignedLong" form="unqualified" />
                    <xs:element name="numero" type="xs:unsignedLong" form="unqualified" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:all>
          </xs:complexType>
        </xs:element>
```

(El documento completo en el **Anexo 9**)

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaCliente>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
        <estado>DF</estado>
        <clave>sr</clave>
        <nombre>Samuel</nombre>
        <paterno>Camacho</paterno>
        <materno>Garcia</materno>
        <razonsocial>Samuel Camacho García</razonsocial>
        <calle>Insurgentes Sur</calle>
        <numero>1325</numero>
        <colonia>San Angel</colonia>
        <municipio>Magdalena Contreras</municipio>
        <cp>01010</cp>
      - <telefonos>
        + <telefono>
        - <telefono>
            <tipodetelefono>compradorcel</tipodetelefono>
            <lada>04455</lada>
            <numero>52202612</numero>
          </telefono>
        + <telefono>
        </telefonos>
      </AltaCliente>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

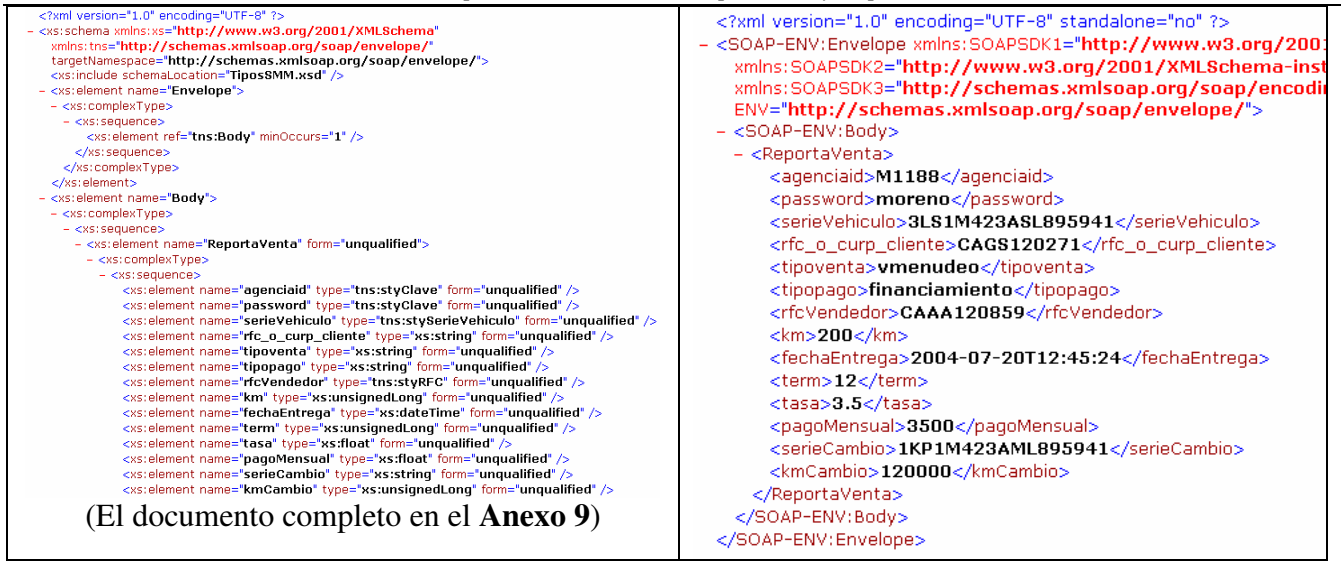


Ilustración 54 XML Schemas VS XML SOAP Requests del Web Service del caso de estudio

De la anterior tabla comparativa se desprenden las siguientes observaciones:

1. Los XML Schemas son documentos XML que describen a toda una familia de documentos XML. Por tanto, se puede decir que los XML Schemas es un lenguaje de descripción de lenguajes, o metalenguaje,
2. Los XML Schemas además de ser el medio formal de descripción de documentos XML, son el punto de partida para validar todos los SOAP Requests que reciben los Java Web services. Validan los tipos de datos simples y compuestos.
3. Los XML Schemas validan el nombre y contenido de cada elemento y la forma en que deben estar ordenados dichos elementos, evaluando si están en una estructura jerárquica o lineal.
4. Los XML Schemas validan tipos de datos derivados definidos por el usuario. En el caso de los Java Web services para órdenes de reparación y reportes de ventas, estos tipos de datos están almacenados en el archivo **TiposSMM.xsd** al que se hace referencia en el elemento include de todos los XML Schemas, pero este punto será tratado con más detalle en la siguiente sección[339].
5. El único namespace que debe manejar el XML Schema es para los sobres de XML SOAP (nombrado con el identificador http://schemas.xmlsoap.org/soap/envelope/), debido a que los parámetros de la transacción fueron definidos como carentes de namespace. Se expresa explícitamente este hecho al asignarle al atributo form el valor de "unqualified" (`form="unqualified"`).
6. En vista de que solamente los XML SOAP Requests, o solicitudes de llamadas a RPC deben ser validados, se tiene un XML SOAP Request para cada uno de los diecinueve tipos de XML SOAP Request del sistema, es decir, los mensajes de entrada de AltaAsesor, AltaCliente, AltaInventario, AltaOrden, AltaUnidad, AltaVendedor, BajaAsesor, BajaCliente, BajaUnidad, BajaVendedor, CancelaOrden, CierraOrden, ImportesOrden, ListaAsesores, ListaClientes, ListaVendedores, ReportaVenta y Transferencia.
7. Todos los XML Schemas que describen a los diecinueve tipos de XML SOAP Requests están expuestos con todo detalle en el **Anexo 9**.

De este ejemplo comparativo se desprenden éstas observaciones, pero las mas importantes es que *para cada tipo de solicitud XML SOAP existe un XML Schema,* que el archivo XSD define el número, orden, contenido y estructura jerárquica de los datos y que hay tipos de datos predefinidos y tipos de datos definidos por el usuario.

[339] **`TiposSMM.xsd`** aloja los tipos de datos definidos por el usuario, por ejemplo los definidos por Saúl Moreno Motte. Si algún programador decide cambiar las reglas de validación de datos, deberá modificar las expresiones regulares contenidas en este archivo o definir su propio archivo XSD.

Como una observación adicional a éste diseño de mensajes, en los documentos de instancia de todos los tipos de XML SOAP Request pudimos haber declarado dos namespaces, uno para los nodos de la gramática SOAP, y otro para los nodos de la gramática de Web services, no obstante, **para simplificar el contenido del mensaje y optimizarlo en tamaño**, sencillamente optamos por no ponerle namespace, asumiendo que esos nodos están destinados para el servicio Web que se hace referencia en el SOAP Action. *Desde luego, se puede exigir que todos los nodos en el Web service contengan un namespace modificando los XML Schemas de los XML SOAP requests correspondientes*.

Otra observación importante es que todos los métodos importan el XSD llamado **TiposSMM.XSD**. ¿Qué es y para qué sirve?

En nuestra interfaz de Java Web services, se tienen dichos tipos de datos definidos por el usuario almacenados en el archivo **TiposSMM.xsd**. A continuación se muestra una reproducción de dicho archivo.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
  - <xs:simpleType name="stySerieVehiculo">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-zA-Z]){17}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styClave">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-zA-Z]){5,}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styRFC">
    - <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z]{3}(.)[0-9]{6}(.){0,10}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styNombre">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([a-zA-Z]|(\s)){5,50}" />
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

Al ver este archivo, se recuerda fácilmente la **sección 2.6** en donde expuse qué era un autómata finito determinístico y cómo se construia a partir de una expresión regular. Aquí se demuestra la utilidad de los XML Schemas, en donde bata con escribir la expresión regular para que se realice la validación.Existen 4 expresiones regulares de mi autoría. Con respecto al tipo de datos que definí como styRFC, originalmente tenía la expresión regular [a-zA-Z]{4}[0-9]{6}(.){0,10}. Eso nos permitía validar RFCs como “MOMS760830MX4” y “MOGT520514KJ4”, pero nos dimos cuenta de que no estabamos aceptando RFCs que también son válidos como “VAV 153040”, “AAN-121014FK3” y “MYM0132314”, por lo que tuvimos que permitir cualquier carácter en la cuarta posición con la expresión regular [a-zA-Z]{3}(.)[0-9]{6}(.){0,10}.

Con respecto a las otras expresiones regulares, se observa que la serie del vehiculo pueden ser 17 caracteres alfanuméricos exclusivamente, sin incluir otro símbolo mas; la clave deben ser al menos 5 caracteres alfanuméricos y el nombre de la persona pueden ser desde 5 hasta 50 caracteres.

Algunas **ventajas que obtuvimos al usar los XML Schemas son las siguientes**:

1. Pudimos validar los campos **fecha**, como por ejemplo, fecha de apertura y fecha de cierre de la orden, con el tipo de datos fecha predefinido en los XML Schemas, por lo que no tuvimos que escribir una expresión regular para validar las fechas. De la misma manera usamos otros tipos de datos como string, integer, long, etc
2. Pudimos validar 4 tipos de **datos definidos por nosotros** (serieVehiculo, clave de la agencia, rfc y nombre) sencillamente con escribir la expresión regular que describe nuestros tipos de datos. Eso nos ahorró escribir y compilar 4 diferentes AFDs para reconocer los tokens correspondientes. El motor de XML Schemas hizo este trabajo por nosotros.
3. Pudimos **corregir el tipo de datos styRFC**, sencillamente corrigiendo o mejorando la expresión regular, sin necesidad de recompilar nada. Solo recargamos el Apache Tomcat para actualizar el XML Schema a memoria.
4. Pudimos hacer altas, bajas, cambios y consultas en los tipos de datos manejados por los XML Schemas sin necesidad de afectar toda la interfaz de Web services, tan solo el módulo implicado en el cambio.
5. Pudimos importar/exportar tipos de datos en un solo archivo, para que solamente se tenga que hacer un solo cambio en el archivo **TiposSMM.xsd**, y no hacer un cambio del mismo tipo de datos en cada uno de los XML Schemas de los métodos (esto es importante porque de otra forma, tendríamos que hacer 19 correcciones de la expresión regular styClave, tan solo para redefinir la regla de validación de ese dato en particular). La instrucción import en los XML Schemas sigue el mismo objetivo que la instrucción import en los lenguajes de programación, a saber, no tener que volver a programar la regla cada vez que la utilicemos.
6. Cabe recordar que si se hubiera seleccionado la tecnología RMI o CORBA para programar estas clases, no se hubieran podido usar los XML Schemas de la misma forma como se hizo aquí.

En el ejemplo manejado en la **sección 4.4** acerca de la embajada americana, podemos decir que el agente consular conoce un número finito de reglas, que a la larga tiende a incrementarse. No obstante, el agente aduanal no sabe si el aspirante a la visa está escribiendo un nombre falso, si el aspirante está declarando contactos no existentes en el extranjero o si sencillamente está mintiendo en sus declaraciones, a pesar de que su solicitud esté bien llenada. De eso se encargan otros agentes consulares quienes aplican entrevistas verbales y discriminan solicitudes en base a otros criterios.

De la misma forma, **a pesar que los XML Schemas validan que el formato, orden, estructura y tipo en los datos sean correctos, desconocen si estos datos pueden insertarse exitosamente en la base de datos**. Existen otras revisiones que realizan las clases de Java programadas en el **Anexo 5.**

Una vez construido e instalado el servicio Web en Java, ¿Cuál es el siguiente paso? De ello se encargará la siguiente sección.

4.5 Utilización de la interfaz de Web Services usando MSSoap Toolkit

Construir un cliente de Web services para los Java Web Services de la Industria Automotriz no tiene que ser una tarea complicada. Un cliente no tiene que ser de difícil construcción, *especialmente si los Java Web Services no lo fueron*. En vista de que uno de los secretos del software exitoso es que sea de fácil utilización, y que rebase las expectativas del cliente, los Web Services deben dar prueba de si mismos de que son una mejor solución que los viejos reportes en formato ASCII enviados por FTP, o que las pantallas del emulador VT3270.

¿Qué se necesita, exactamente, para usar sin problemas los XML Web services? Tal como un cocinero reune los ingredientes de su próxima creación gastronómica, nosotros debemos reunir los siguientes elementos para el cliente:

- Una base de datos relacional local (de preferencia pequeña).
- Un lenguaje de programación amigable con la pequeña base de datos relacional local.
- Una biblioteca que convierta documentos XML en objetos (marshall) y viceversa (unmarshall). La gramática de los documentos XML debe ser SOAP.
- Una biblioteca que pueda enviar documentos XML a algún servidor Web usando el protocolo HTTP.

Las modernas PCs regularmente precargadas con WindowsXP y Microsoft Office 2003 no tienen inconveniente de cumplir con estos sencillos requerimientos de la siguiente manera:

- Microsoft Access ofrece un motor de base de datos para archivos pequeños (en formato MDB) sin descuidar las relaciones que proveen las reglas de integridad referencial.
- Microsoft Access ofrece dentro de su motor de base de datos la capacidad de almacenar código en módulos dentro de la misma base de datos, y el código contiene diversas versiones de librerías para acceder adecuadamente a los datos: Data Access Objects (DAO) y ADO (Active Data Objects)
- Librería Microsoft SOAP Toolkit 3.0. Esta librería –De distribución gratuita en Internet- realiza tanto la elaboración como la transferencia de XML vía HTTP al servidor que se le especifique mediante su URL. Esta librería mide en disco 3,768,320 bytes, y puede descargarse directamente del sitio de Microsoft si desea compararse con la provista en el CD Toolkit que acompaña a la presente tesis. Microsoft SOAP Toolkit es 100% compatible con Microsoft Access como con Windows XP.

Los servidores y terminales –basados en sistemas operativos unix, o más recientemente, en linux-, tienen muchas formas de cumplir con los requerimientos para el cliente. Por citar un caso:

- Existen múltiples motores de bases de datos para UNIX o LINUX. Se pueden usar desde los motores de base de datos precargados en el sistema –Posgres, MySQL- hasta los motores de base de datos relacionales que estén disponibles en la compañía –Oracle8, Oracle9i, Sybase, etc-
- Los sistemas operativos UNIX o LINUX contienen múltiples lenguajes de programación. Se tienen intérpretes para PERL, Python, C++ (gcc) y si se desea, se pueden descargar nuevos lenguajes de programación gratuitos de la red, una buena opción, Java.
- La industria ha fabricado Librerías para enviar SOAP vía HTTP adaptable a todo tipo de lenguaje. Existen librerías para PERL, Python, C++ (XercesC), Java(Xerces for Java, JAXP, JAXM), etc. Conseguir una librería gratuita de SOAP para nuestro lenguaje de programación favorito está a un click de distancia.

Aunque es opcional el hecho de que el cliente puede guardar los datos de las transacciones que efectúa, es recomendable que lo haga. En tal caso, puede soportar sus operaciones con una base de datos relacional. En el **Anexo 12** se provee un diagrama Entidad-Relación para soportar los datos del programa cliente –En este caso los datos del distribuidor-. Como se comentó tempranamente en este trabajo de tesis, la diferencia que existe con el diagrama Entidad-Relacion de la base de datos en el corporativo radica en el hecho de que la base de datos soportará únicamente los datos de una sola distribuidora, por lo que no necesitará la tabla tblAgencias ni las claves agenciaid relacionadas.

En la industria automotriz, como es de esperarse, se desarrollaron clientes para el Web Service de diversos tipos. *Antes de que cada proveedor certificado de software en las agencias distribuidoras eligiera el motor, el lenguaje y la librería de su preferencia, el departamento de enlace al distribuidor debía poner el ejemplo al desarrollar una aplicación que mostrara el uso de los Web services*. Este desarrollo, conocido por todos los distribuidores como "la plantilla", consta de un archivo de Microsoft Access que contiene la definición de base de datos, la librería de Microsoft Soap Toolkit, y rutinas de Visual Basic for Applications para hacer el Proxy entre el cliente de Web service y el Java Web Service.

En Microsoft Visual Basic For Applications, ¿Cómo se definió la base de datos del Cliente en Microsoft Access? Tomando como lineamiento el **Anexo 12** del diagrama entidad relación del cliente, generé el código del **Anexo 13** para construir una pequeña base de datos MDB en Visual Basic for Applications. En éste código se muestra como crear las estructuras y como insertar los datos en los catálogos, datos que serán enviados a los Web services.

El código del **Anexo 13** muestra que, además de definir los campos del diagrama entidad relación, definimos campos tales como mAltaAsesor y dtAltaAsesor. Estos campos tienen el propósito de almacenar el mensaje que resulta de hacer una transacción, y la fecha en la cual se generó la transacción. En este caso, se tiene un par de estos campos (m, dt) para cada método de web services, con el fin de llevar un registro que compruebe si una transacción se efectuó o no, y la fecha como acuse de recibo.

Al ejecutar las rutinas Regenera y AlimentaCatálogos, Microsoft Access, genera las tablas, llaves, restricciones y valores en catálogo. Adicionalmente elabora un diagrama de relaciones que es equivalente al diagrama Entidad-Relación del **Anexo 12**. Aquí se muestra una reproducción de dicho diagrama:

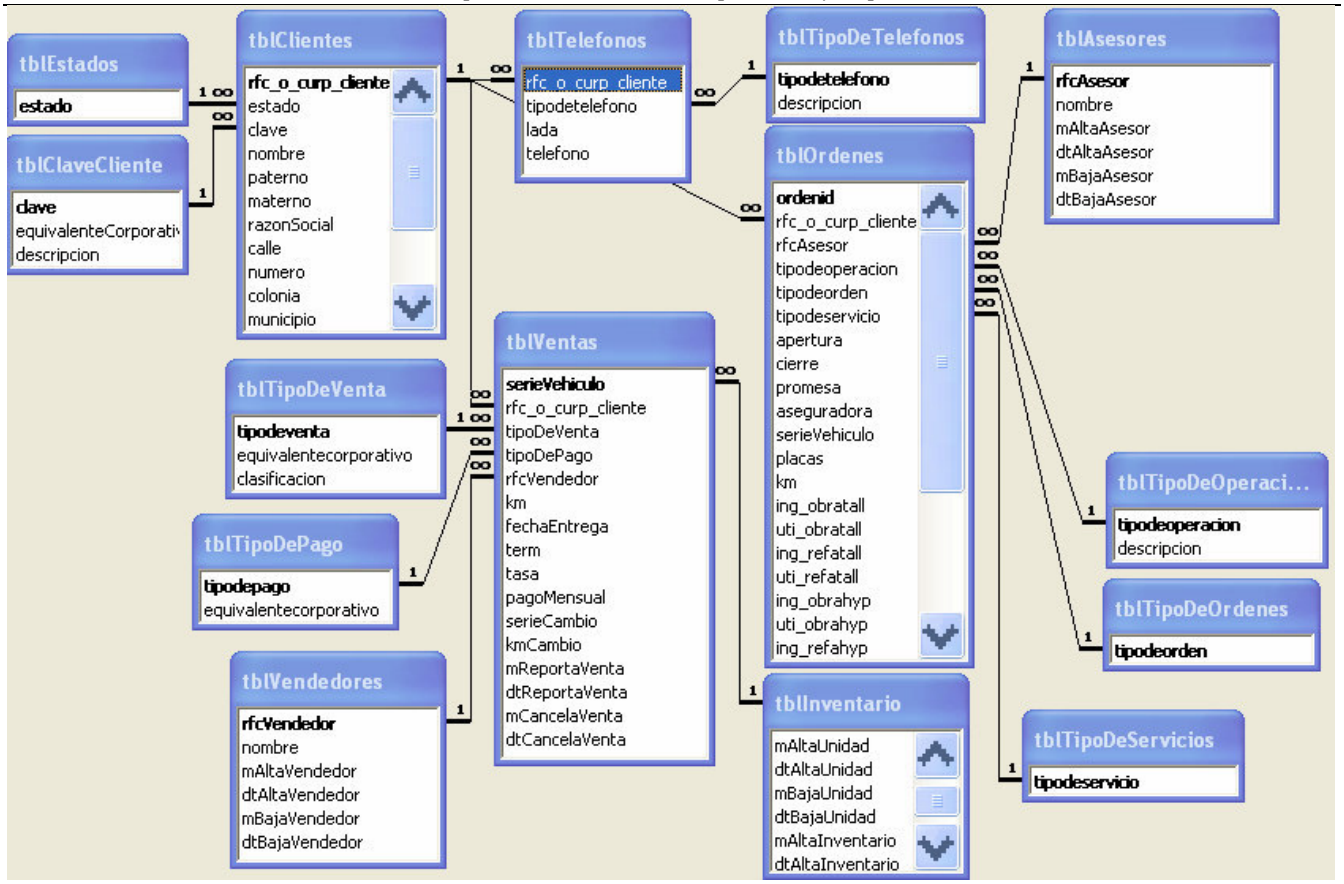


Ilustración 55 Diagrama Entidad-Relación del cliente generado automáticamente en MSAccess

Teniendo la base de datos en formato MDB, así como la librería MSSOAP Toolkit 3.0 Instalada en WindowsXP[340] ¿Cómo pueden utilizarse los Web Services? En el **Anexo 14** se tienen las rutinas que establecen el Proxy de los *diecinueve métodos de Web Services. En la metodología del capítulo 4 se definió que la arquitectura RPC busca establecer proxies, para que un componente remoto se vea como una función local*. Pues bien, el **Anexo 14** define diecinueve "funciones locales", al construir sobres de XML con los parámetros necesarios en el orden correcto, y enviarlos por HTTP.

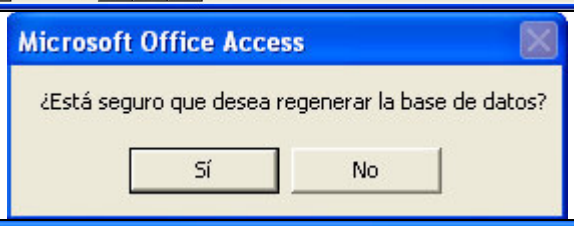
Este Proxy permite que un conjunto de funciones o métodos cargados en la memoria de una Java Virtual Machine, puedan ejecutarse remotamente, desde un cliente de Microsoft Access, programado con Visual Basic For Applications. No importa cuántos routers o firewalls existen entre el cliente y el servidor, ya que la conexión HTTP se encarga de que los mensajes sean transferidos de un punto a otro sin bloqueos.

[340] Para instalar las librerías y la aplicación completa, véase el **Anexo 06** "*Guía de Instalación de la Interfaz Java Web Services*"

¿Cómo funciona el Cliente de Web Services? Al abrir el archivo de Microsoft Access, WSCLIENT.MDB, se muestra una carátula con el nombre del autor y dos botones, continuar y salir. He aquí la ventana:

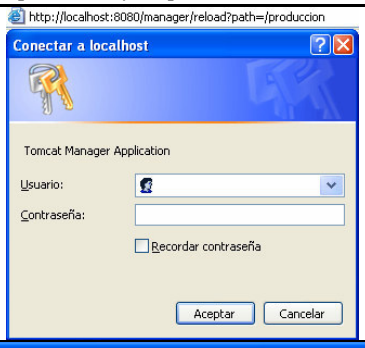
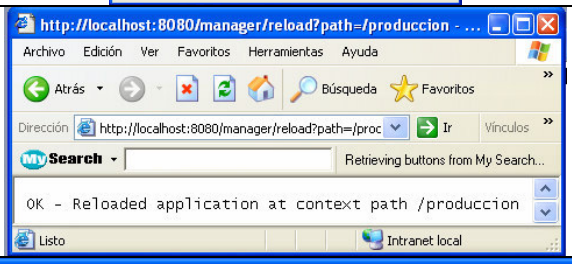
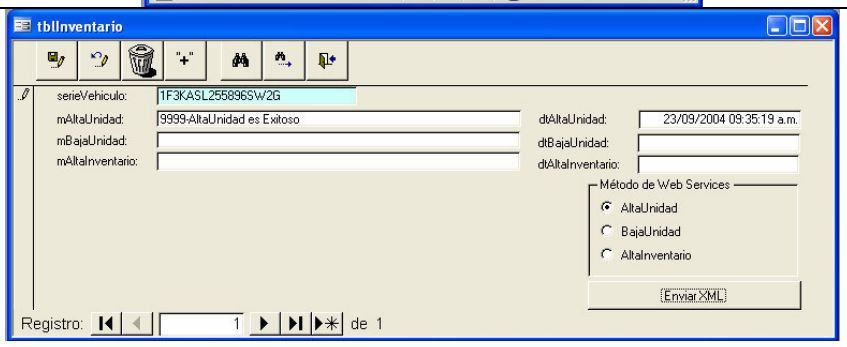
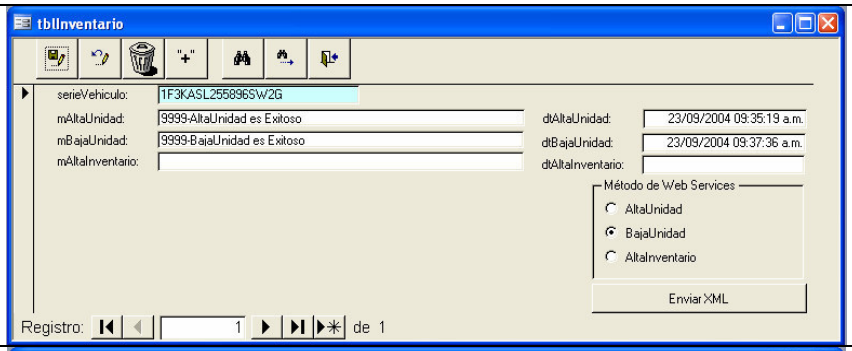
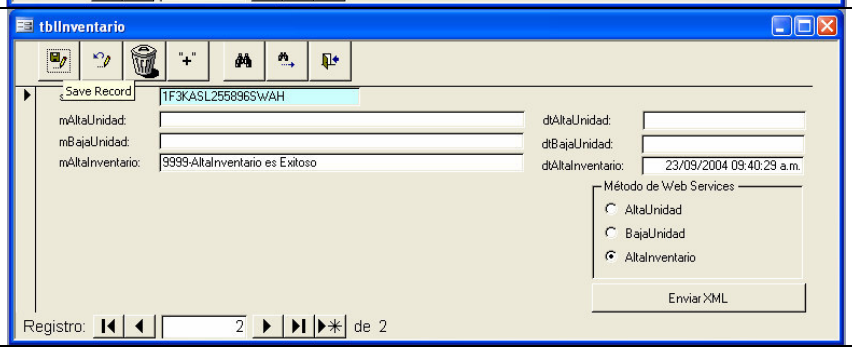
Al dar click en Continuar, se muestra otra pantalla con el menú principal del cliente. Se observa aquí que se tienen las cuatro operaciones principales del sistema: Inventario, Transferencias, Ventas y Órdenes de reparación. Adicionalmente, se tiene la utilería para regenerar las tablas y relaciones de la base de datos, así como modificar los parámetros del sistema.

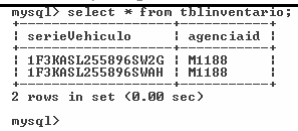
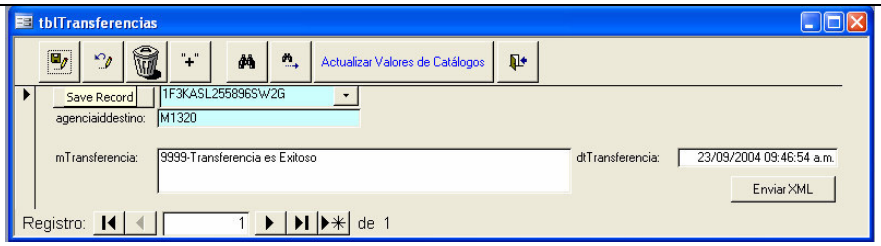
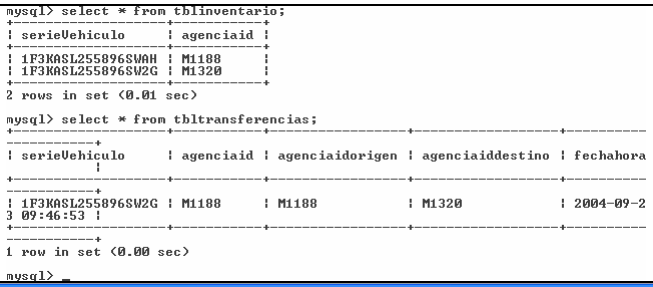
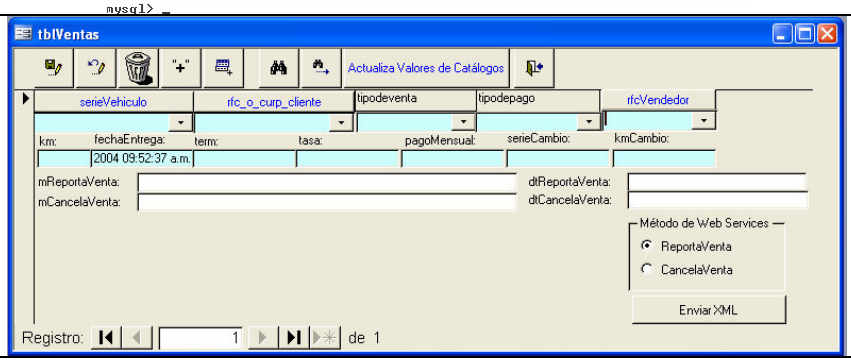
¿Qué ocurre si se selecciona la opción "Regenerar tablas y relaciones"? En tal caso aparece una ventana solicitando la confirmación de la operación. Si se regeneran las estructuras se pierde de forma irreversible los datos, a menos que se cuente con un respaldo del MDB.



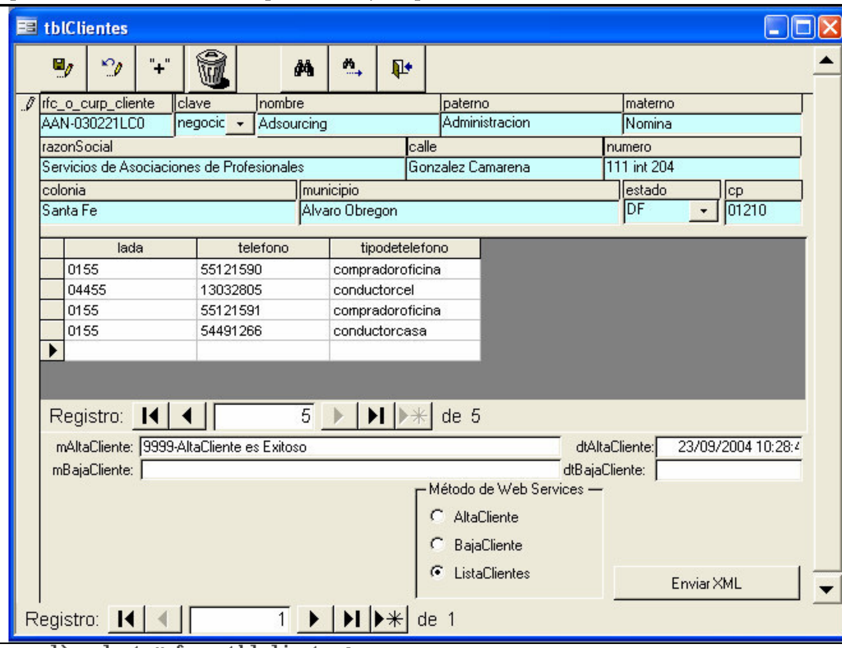
Si se contesta "No" a la pregunta anterior, el cliente de Web Services cancelará la operación y respetará las estructuras y los datos existentes.

Al seleccionar la segunda opción del menú principal, aparecerá una ventana con tres parámetros centrales: el usuario o el identificador de la agencia, el password, y un url que le dice al cliente en donde se encuentra el servidor web de páginas y aplicaciones Tomcat. En la parte superior se tiene una barra de herramientas

Antes de realizar cualquier transacción, es necesario iniciar el Tomcat Application Server, y reiniciar la aplicación. El Application Manager está almacenado en el path y el comando reload recarga la aplicación. Antes de ejecutarse el comando reload, se preguntará el usuario y la contraseña del servidor. Se definió en este caso admin/admin. No olvidar presionar el botón Aceptar.	
Se debe esperar pacientemente hasta que el Tomcat Application Server muestre un mensaje notificando al usuario sobre la aplicación que ha sido recargada en memoria.	
Al ingresar a la opción "Inventario de Vehiculos Nuevos" se muestra una caja de texto en donde se puede insertar el número de serie del vehiculo. En este caso, se capturó un número de serie vehicular de 17 posiciones. Aquí se muestra cómo se manda a llamar al método AltaUnidad usando la función de Proxy correspondiente del **Anexo 14**. Si no se generó error, se muestra el mensaje 9999-AltaUnidad es Exitoso, y se coloca la fecha y la hora en la caja de texto asociada. Esta forma fue construida con el Forms Wizard de MSAccess, por lo que se tiene un manejo directo de la base de datos local y un manejo remoto de la base de datos remota.	
Una unidad puede ser dada de baja con la misma facilidad con la que se dio de alta. En este caso, se selecciona el botón de opción BajaUnidad y se envía el XML al Web service. Después de algunos segundos, el sistema notificará que la unidad ha sido satisfactoriamente eliminada de la tabla de inventario remota.	
Ahora, suponiendo que se agrega otro número de serie al inventario del cliente, se tendrán dos unidades en el inventario local (con terminación 2G y AH). En este caso en vez de enviar dos XML de AltaUnidad, se puede enviar un solo XML de AltaInventario, con la información de las dos unidades. En este caso, se selecciona el radio button AltaInventario y se presiona el comando "Enviar XML". Después de unos segundos, se mostrará una clave 9999 indicando que la transacción fue exitosa.	

Si se consulta la base de datos remotamente, se comprobará que las dos unidades ya están en el inventario, y están asociadas a la agencia M1188, en conformidad con el dato del catálogo de parámetros del cliente.	
En la tabla de inventario, se requiere guardar los datos en la base de datos local antes de tratar de enviarlos al Web service. Esto se puede hacer con el botón guardar que se encuentra en la barra de herramientas de la ventana inventario, marcada con el dibujo de un diskete. De otra forma, si no se guardan los datos antes de enviarlos, se mostrará la siguiente advertencia en el cliente:	
La siguiente opción del menú principal está dedicada a las transferencias de vehículos. Por ejemplo, si se desea enviar la unidad con terminación 2G a la distribuidora M1320, debe seleccionarse en el menú de cortina, escribir la clave de la agencia destino, guardar la información y presionar "Enviar XML". Después de aguardar unos segundos, se mostrará de nueva cuenta el mensaje 9999, que indica que la transferencia fue efectuada correctamente en el sistema.	
Al consultar la base de datos remota, se observará que en el inventario la unidad ha cambiado de dueño, y también se observa que un nuevo movimiento ha sido dado de alta en la tabla tbltransferencias, lo que es útil para llevar un control exacto sobre que unidad se transfirió a qué agencia, y quién fue el responsable de dicha operación, y la fecha y la hora en que se efectuó la operación.	
La tercer sección fue diseñada para enviar la información de las ventas. Después de oprimir el botón "+", para agregar un nuevo registro, aparecerá una pantalla con los campos en blanco. Antes de empezar a llenar los campos, se requiere alimentar los catálogos de clientes y de vendedores. En el campo serieVehiculo ya se tiene un numero de serie que se puede utilizar sin problemas.	

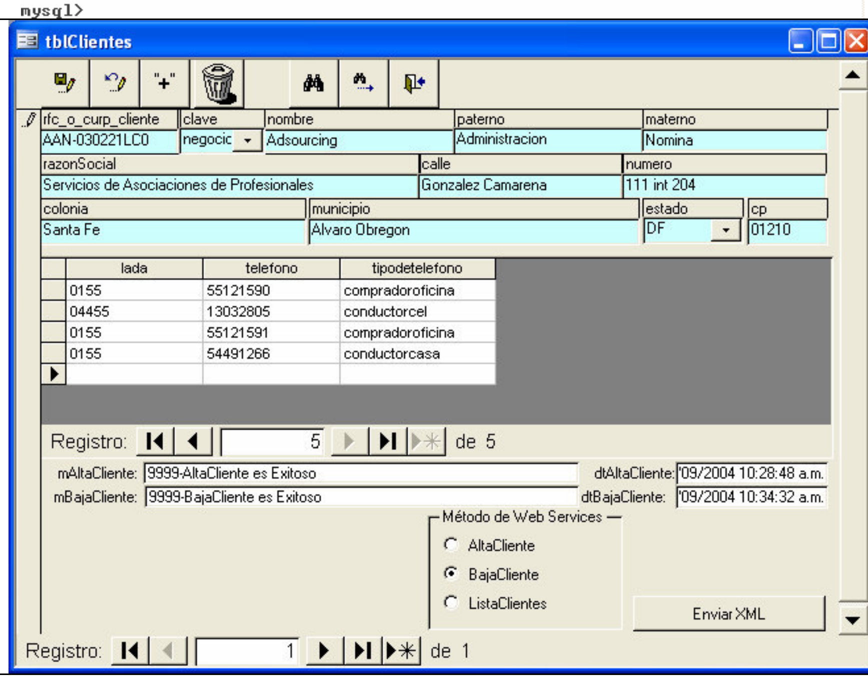
Al hacer clic en el botón rfc_o_curp_cliente de letras azules de la ventana anterior, el sistema abre una ventana para capturar los datos del cliente. Aquí se capturan unos datos de ejemplo, insertando además cuatro teléfonos: dos de oficina, uno de casa y un celular.
Una vez capturados los datos, el distribuidor guarda su información con el botón guardar marcado con un disquete y un lápiz. A continuación se selecciona el método AltaCliente y el distribuidor debe oprimir el botón “Enviar XML”.
Tal como en los otros métodos, el sistema de Web Services responderá con un 9999 y el cliente de Web services insertará la fecha hora locales.



Al inspeccionar la base de datos remota, se verificó que el registro del cliente fue adecuadamente insertado.

```
mysql> select * from tblclientes;
+--------------------+-----------+--------+----------+------------+---------------+
---+----------+--------------------------------------------+-----------------------+-
-----------+----------+------------------+--------+
| rfc_o_curp_cliente | agenciaid | estado | clave    | nombre     | paterno
  | materno | razonSocial                                | calle                 |
numero      | colonia  | municipio      | cp    |
+--------------------+-----------+--------+----------+------------+---------------+
---+----------+--------------------------------------------+-----------------------+-
-----------+----------+------------------+--------+
| AAN-030221LC0      | M1188     | DF     | negocios | Adsourcing | Administraci
on | Nomina   | Servicios de Asociaciones de Profesionales | Gonzalez Camarena |
111 int 204 | Santa Fe | Alvaro Obregon | 01210 |
+--------------------+-----------+--------+----------+------------+---------------+
---+----------+--------------------------------------------+-----------------------+-
-----------+----------+------------------+--------+
1 row in set (0.39 sec)

mysql>
```

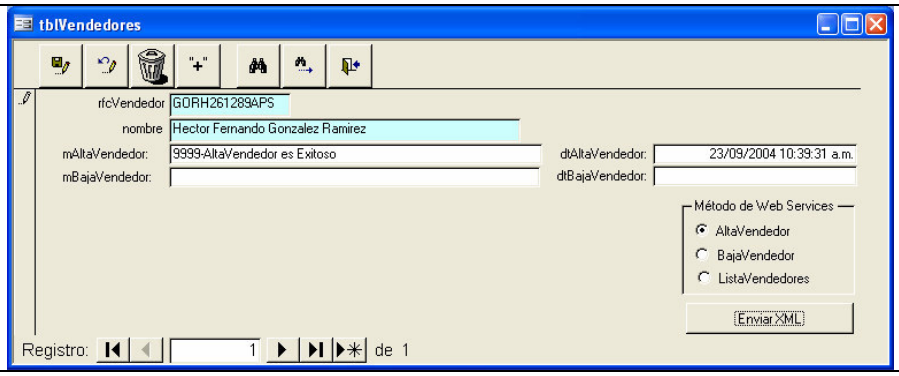
Por otra parte, es posible dar de baja un cliente, en caso de que se desee corregir un dato. En este caso, se debe dar de baja con el método BajaCliente, hacer las correcciones de forma local y volver a utilizar elmétodo AltaCliente. El método BajaCliente también muestra un 9999 si la operación de eliminación fue existosa sobre la base de datos remota. Esta imagen es un ejemplo de ello:



Si se desea saber cuantos clientes se tiene en la base de datos remota, bastará con seleccionar el botón de radio "ListaClientes" y "Enviar XML". El método correspondiente en el Proxy enviará el nombre del método, el usuario y la contraseña. En cambio, el servidor devolverá un XML con los datos solicitados en el siguiente estilo:

```
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ListaClientesRespuesta xmlns:m="http://www.unam.mx/ListaClientesRespuesta">
      - <cliente>
          <rfc_o_curp_cliente>AAN-030221LC0</rfc_o_curp_cliente>
          <estado>DF</estado>
          <clave>negocios</clave>
          <nombre>Adsourcing</nombre>
          <paterno>Administracion</paterno>
          <materno>Nomina</materno>
          <razonsocial>Servicios de Asociaciones de Profesionales</razonsocial>
          <calle>Gonzalez Camarena</calle>
          <numero>111 int 204</numero>
          <colonia>Santa Fe</colonia>
          <municipio>Alvaro Obregon</municipio>
          <cp>01210</cp>
        - <telefonos>
          - <telefono>
              <tipodetelefono>compradoroficina</tipodetelefono>
              <lada>0155</lada>
              <numero>55121590</numero>
            </telefono>
          + <telefono>
          + <telefono>
          + <telefono>
          </telefonos>
        </cliente>
      </m:ListaClientesRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

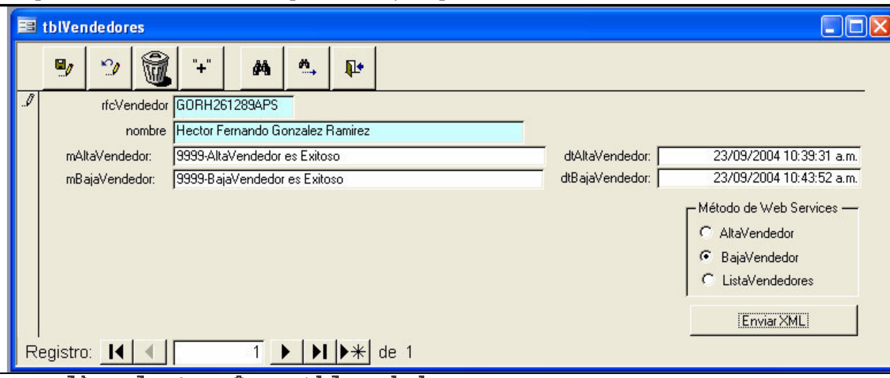
De forma análoga al catálogo de clientes, para reportar las ventas se tiene un catálogo de vendedores. En la lámina se muestra cómo fue capturada la información de uno de tales vendedores, y en este caso ejecuté el método AltaVendedor para ingresar la información en la base de datos remota. El servidor, como en las veces anteiores, no mostró objeción con respecto a estos datos, y después de un pequeño lapso de tiempo, genera el mensaje 9999. El cliente coloca la fecha y hora local en el sistema.



Si es seleccionado el método de Web services ListaVendedores, se enviará la solicitud de información con el nombre del método, el usuario y el password. El Web service remoto devolverá un documento XML, un tanto más simple con respecto al método "ListaClientes", y este método se encargará de reunir la información de todos los distribuidores para la agencia M1188, que es con la que trabaja el catálogo. Se abrirá nuevamente una ventana de navegador y mostrará la estructura de XML de la siguiente manera.

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ListaVendedoresRespuesta
        xmlns:m="http://www.unam.mx/ListaVendedoresRespuesta">
      - <vendedor>
          <rfcVendedor>GORH261289APS</rfcVendedor>
          <nombre>Hector Fernando Gonzalez Ramirez</nombre>
        </vendedor>
      </m:ListaVendedoresRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

El método BajaVendedor eliminará el dato solicitado en la tabla remota tblVendedores y devolverá un mensaje 9999. Debido a que se necesitará este dato para futuro uso, volví a utilizar el método AltaVendedor.

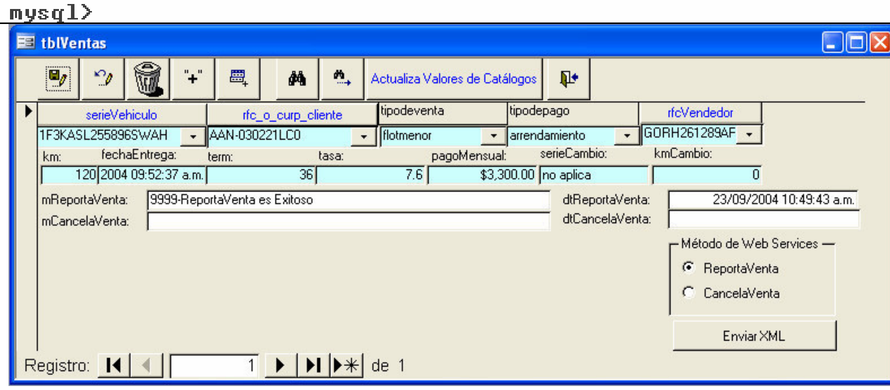


Para éste método, también verifique que la tabla vendedores estuviera almacenando adecuadamente los datos de los vendedores.

```
mysql> select * from tblvendedores;
+---------------+------------+----------------------------------+
| rfcVendedor   | agenciaid  | nombre                           |
+---------------+------------+----------------------------------+
| GORH261289APS | M1188      | Hector Fernando Gonzalez Ramirez |
+---------------+------------+----------------------------------+
1 row in set (0.20 sec)

mysql>
```

Al tener valores en los catálogos, noté que se requiere actualizar los catálogos manualmente, debido a que Microsoft Access no lo hace automáticamente. Por tanto, generé un botón de comando para actualizar los datos en los controles de combinación para selección de valores.

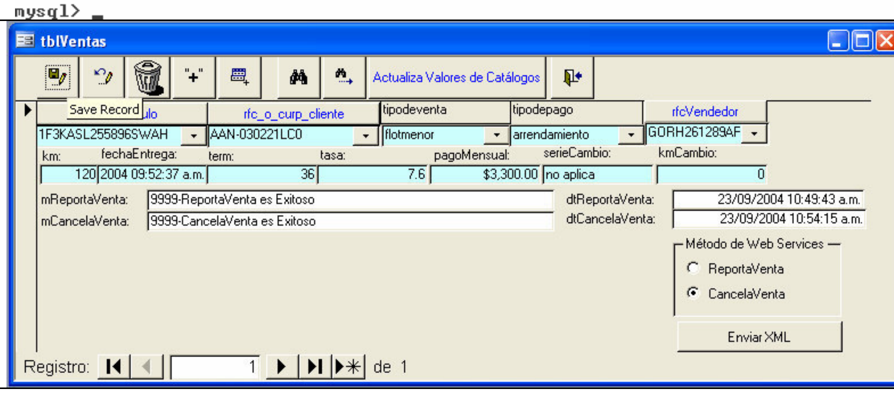
Después de actualizar los valores de catálogos, capturé todo el registro, especificando el número de serie, el rfc del cliente, el tipo de venta, tipo de pago, rfc del vendedor, el kilometraje de la unidad, y algunos datos adicionales mas. Después de usar el botón "Guardar Cambios" marcado con un disquete, seleccioné el botón de radio ReportaVenta y "EnviarXML". Al esperar unos segundos el Web service devolvió un 9999, y después volví a guardar los cambios con el botón guardar.



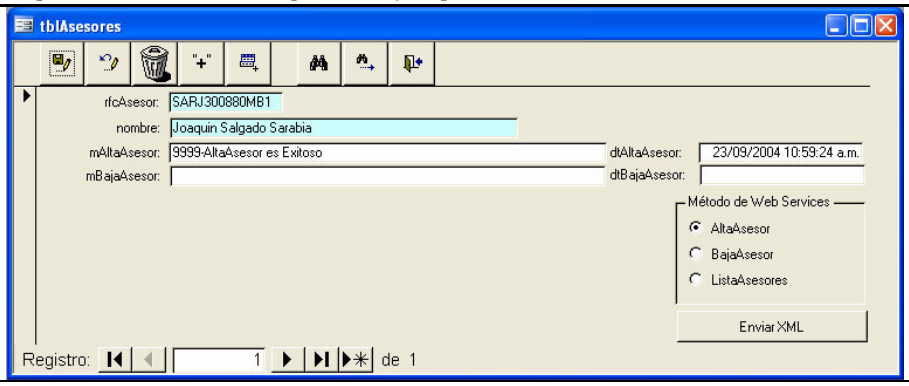
En la base de datos del corporativo revisé que los datos estuvieran correctamente insertados. He aquí el resultado esperado:

```
+-------------------+---------------------+------------+-------------+------------
---+---------------+------+---------------------+------+------+-------------+---
-----------+----------+
| serieVehiculo     | rfc_o_curp_cliente | agenciaid | tipodeventa | tipodepago
   | rfcVendedor   | km   | fechaEntrega        | term | tasa | pagoMensual | se
rieCambio | kmCambio |
+-------------------+---------------------+------------+-------------+------------
---+---------------+------+---------------------+------+------+-------------+---
-----------+----------+
| 1F3KASL255896SWAH | AAN-030221LC0      | M1188     | flotmenor   | arrendamien
to | GORH261289APS |  120 | 2004-09-23 09:52:37 |   36 | 7.60 |     3300.00 | no
 aplica    |        0 |
+-------------------+---------------------+------------+-------------+------------
---+---------------+------+---------------------+------+------+-------------+---
-----------+----------+
1 row in set (0.05 sec)

mysql> _
```

Si es necesario eliminar el reporte de una venta de un vehículo nuevo, el método de Java Web Services diseñado para ello es CancelaVenta. Aquí se muestra el resultado de haberlo usado. Obviamente, el método toma los datos actuales en la pantalla. En este caso, el método devolvió un estatus 9999 lo que indica que la operación de borrado fue exitosa.



Por último, la opción de órdenes del menú principal lleva a una pantalla que tiene como catálogos los datos del cliente y los datos del asesor, de una forma análoga como se hace en los reportes de ventas. En vista de que ya se describió el funcionamiento del catálogo de clientes, deseo mostrar directamente el catálogo de asesores, que es completamente análogo al catálogo de vendedores. He aquí un dato de un RFC y de un nombre. En este dato ya se aplicó el método AltaAsesor, con resultados positivos (se obtuvo un estatus 9999 como resultado de esa transacción)



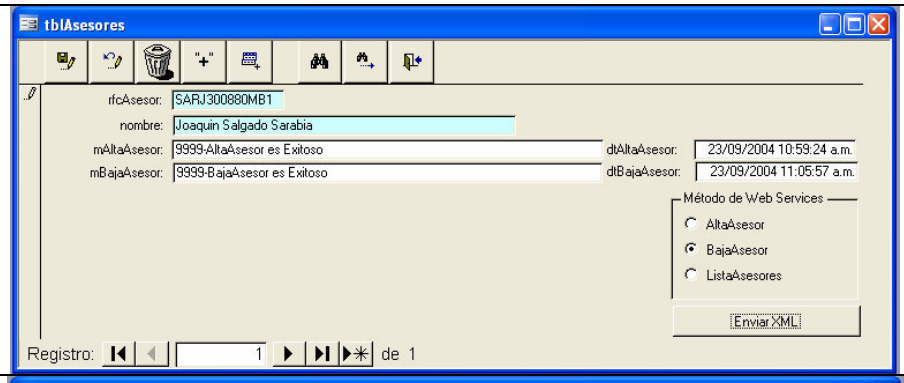
Si se utiliza el método ListaAsesores, el Web service devolverá un XML con todos los asesores encontrados para la agencia distribuidora M1188, y el navegador los mostrará en el formato jerárquico ya conocido. Este archivo XML puede explotarse de la forma como mejor parezca al proveedor de sistemas certificado.

```
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ListaAsesoresRespuesta xmlns:m="http://www.unam.mx/ListaAsesoresRespuesta">
      - <asesor>
          <rfcAsesor>SARJ300880MB1</rfcAsesor>
          <nombre>Joaquin Salgado Sarabia</nombre>
        </asesor>
      </m:ListaAsesoresRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

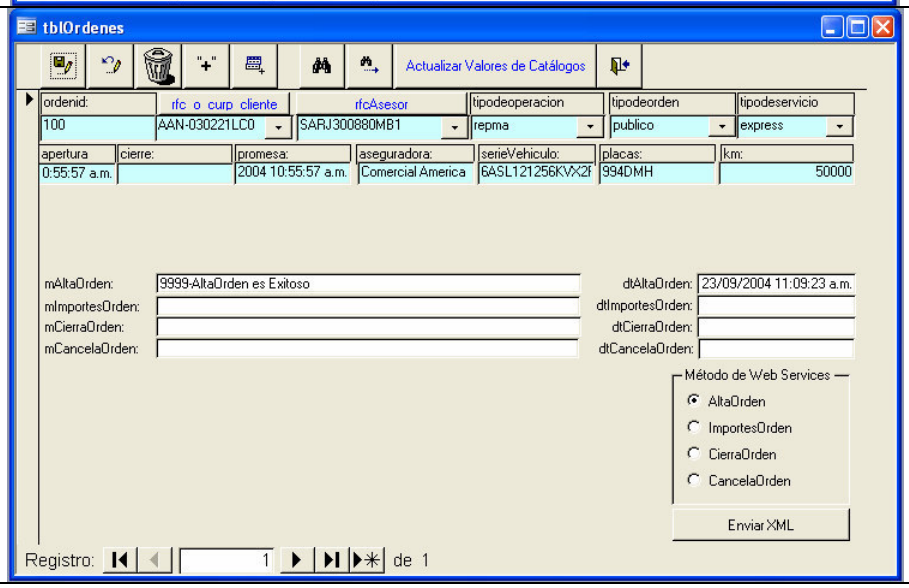
La información recibida en el XML es exactamente la misma que se tiene en el servidor de base de datos de MySQL en el corporativo. Aquí se reproduce una consulta hecha directamente en el servidor.

```
mysql> select * from tblAsesores;
+--------------+------------+--------------------------+
| rfcAsesor    | agenciaid  | nombre                   |
+--------------+------------+--------------------------+
| SARJ300880MB1 | M1188     | Joaquin Salgado Sarabia  |
+--------------+------------+--------------------------+
1 row in set (0.17 sec)

mysql>
```

Si la información de un Asesor deja de ser útil para la distribuidora, sencillamente ese asesor debe ser dado de baja, para que ya no pueda utilizarse su RFC para elaborar más órdenes.El método BajaAsesor se encarga de esta tarea y devolverá un 9999 al no existir problemas en la operación.



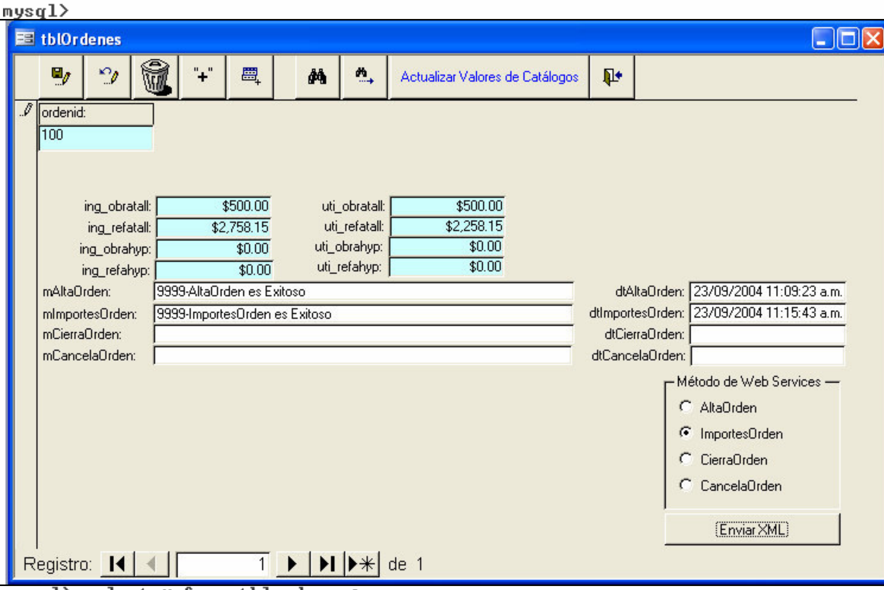
Al tener los catálogos actualizados, actualize los valores de catálogos presionando el botón correspondiente. Después capturé todos los datos solicitados excepto por la fecha de cierre, la cual puedo especificar más adelante. Guardé la información con el botón guardar (marcado con un disquete) y procedí a enviar el Alta de la Orden con el botón de radio y el botón de comando "Enviar XML". El servidor de Web mostró el mensaje "9999-AltaOrden es Exitoso"



Tan solo como una medida preventiva, me cercioré de que la información existiera en la base de datos de MySQL. He aquí el resultado de mi consulta:

```
mysql> select * from tblordenes;
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
| ordenid | agenciaid | rfc_o_curp_cliente | rfcAsesor    | tipodeoperacion | t
ipodeorden | tipodeservicio | apertura             | cierre               | promes
a            | aseguradora       | serieVehiculo     | placas | km     | ing_obr
atall | uti_obratall | ing_refatall | uti_refatall | ing_obrahyp | uti_obrahyp |
 ing_refahyp | uti_refahyp |
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
| 100     | M1188     | AAN-030221LC0     | SARJ300880MB1 | repma          | p
ublico     | express        | 2004-09-23 10:55:57 | 0000-00-00 00:00:00 | 2004-0
9-23 10:55:57 | Comercial America | 6ASL121256KVX2PH8 | 994DMH | 50000 |
 NULL |          NULL |          NULL |          NULL |         NULL |         NULL |
       NULL |        NULL |
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
1 row in set (0.09 sec)

mysql>
```

Opcionalmente, se puede reportar los importes asociados a esa orden, es decir, los ingresos y/o utilidades asociadas con la orden en cuestión. Después de capturarlos y guardarlos con el botón de la barra de herramientas correspondiente, envié el XML de ImportesOrden. El sistema de Web services respondió con el mensaje 9999.



A fin de revisar los cambios en la base de datos asociada con el Web service, volví a ejecutar la consulta sobre la tabla tblordenes, la cual ya reflejaba los cambios en la base. Es pertinente destacar que los campos que anteriormente habían sido Null, ahora tenían valores numéricos, con 0.00 en el peor de los casos.

```
mysql> select * from tblordenes;
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
| ordenid | agenciaid | rfc_o_curp_cliente | rfcAsesor    | tipodeoperacion | t
ipodeorden | tipodeservicio | apertura             | cierre               | promes
a            | aseguradora       | serieVehiculo     | placas | km     | ing_obr
atall | uti_obratall | ing_refatall | uti_refatall | ing_obrahyp | uti_obrahyp |
 ing_refahyp | uti_refahyp |
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
| 100     | M1188     | AAN-030221LC0     | SARJ300880MB1 | repma          | p
ublico     | express        | 2004-09-23 10:55:57 | 0000-00-00 00:00:00 | 2004-0
9-23 10:55:57 | Comercial America | 6ASL121256KVX2PH8 | 994DMH | 50000 |       5
00.00 |       500.00 |      2758.15 |      2258.15 |        0.00 |        0.00 |
        0.00 |        0.00 |
+---------+-----------+-------------------+--------------+-----------------+--
----------+---------------+------------------------+----------------------+-------
------------+-------------------+-------------------+---------+--------+--------
-----+---------------+------------------+-----------------+----------------+
-------------+-------------+
1 row in set (0.00 sec)

mysql> _
```

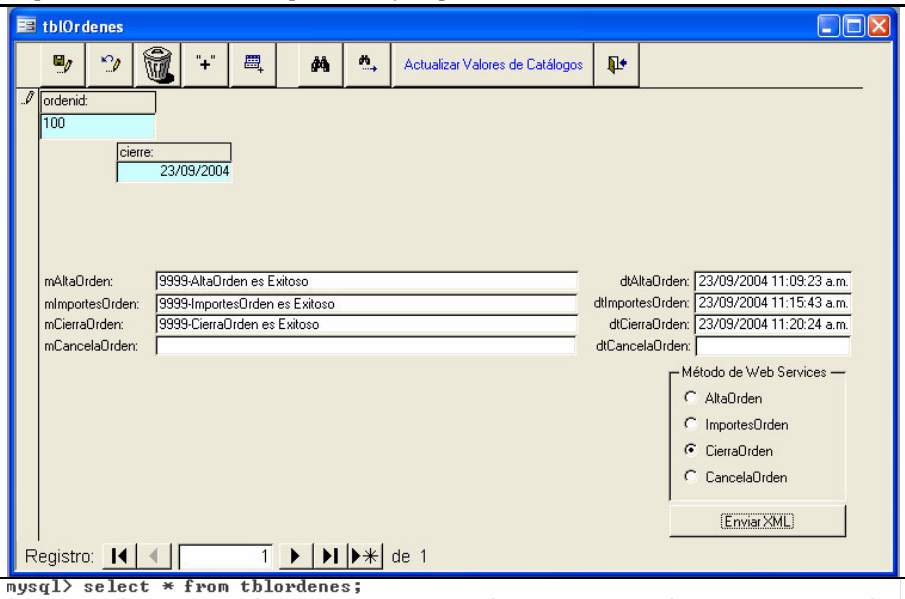
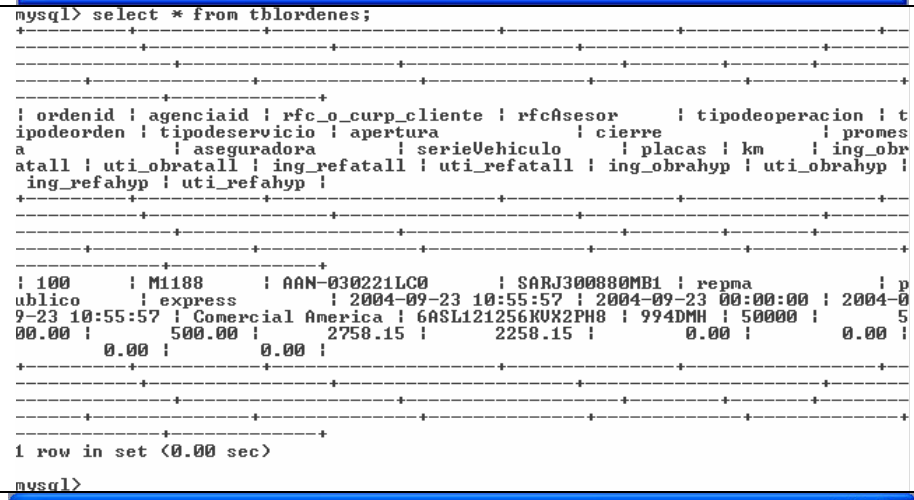
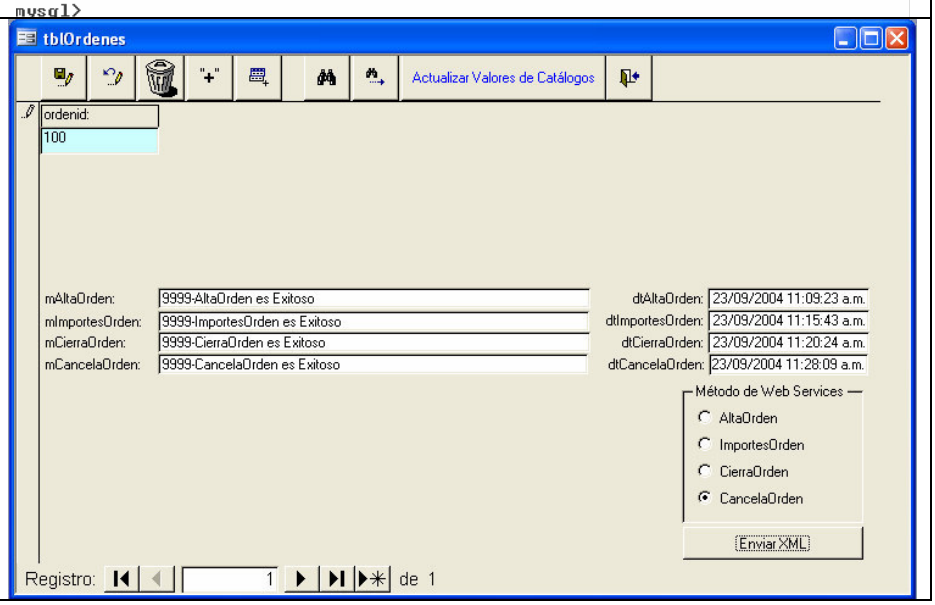
Al escoger el Método de Web Services Cierra Orden, oculto todos los campos y solo muestro el número de orden y la fecha de cierre. Una vez capturada, guardé el registro y utilizé el método de Web services CierraOrden. La actualización a la base de datos en representación del cliente, hecha por el servidor, fue exitosa, y la aplicación devolvió un mensaje 9999 acerca de CierraOrden	
¿Qué cambio se generó en la base de datos? El campo que antes guardaba el valor 0000-00-00 00:00:00 ahora ya tenía el valor coherente de 2004-09-23 10:55:57. Esta órden, al estar cerrada, será enviada por el corporativo al área de estadísticas y encuestas para su procesamiento por el proveedor correspondiente.	
El método de Web services CancelaOrden solamente requiere como parámetro el número de orden. Aquí, cancelar la orden 100 no representó ningún problema, y el método CancelaOrden también devolvió un 100.	

Tabla 32 Uso de los métodos deWeb services desde el cliente de MSAccess

Éste cliente, en formato MDB, es que el corporativo desarrolló para mostrar a los Distribuidores el funcionamiento de los Web services, mas no fue la interfaz que usaron sino hasta Marzo de 2003. A partir de esa fecha el distribuidor siguió utilizando los servicios Web de Ventas y Servicio, usando nuevos programas cliente, en algunos casos, basados nuevamente en Microsoft SOAP Toolkit.

En otros casos, los clientes fueron programados en Xerces(C++), Java Applets y JAXP, y otro tipo de librerías. Esto probó la funcionalidad universal que provee el servicio. De hecho, nuestros Web services fueron pensados para proveer solamente la funcionalidad de un servicio transaccional, permitiendo que el cliente decida que "front end" o qué aspecto gráfico/aplicativo tendrá su aplicación. En éste sentido, el Web service puede ejecutarse desde un Navegador, desde una subrutina en una aplicación local, como una aplicación en línea de comandos, como dos aplicaciones transfiriendo información en un esquema Business to Business, etc.

4.5.1 Errores más comunes en el uso de los Web services

En la sección anterior se mostró el funcionamiento normal de los clientes de Web services. No obstante, debido a que en todos los sistemas, en ocasiones se generan errores, se deben tomar acciones tanto preventivas como correctivas para tener estas situaciones controladas.

Un caso es el no poder abrir el puerto 80 en el firewall del distribuidor, debido a políticas de firewall internas. En este caso, quizas se desee utilizar el puerto 8080, debido a que los Java Web services están escuchando continuamente por ambos puertos, como se muestra en el **Anexo 6** "Guía de Instalación del Sistema". En el caso de que no encuentre el servidor, el cliente mostrará un mensaje como el siguiente:

Ilustración 56 Mensaje de error por bloqueo del puerto 80 u 8080

La solución para este caso, es que el distribuidor puede abrir su catálogo de parámetros y probar con el puerto 80 y el 8080. Cuando se usa el puerto 80, pueden omitirse los dos puntos y el 80 después del nombre del host, como se ve en la siguiente figura:

Ilustración 57 Modificación del URL para utilizar el puerto 80 u 8080

Otro problema en el que comúnmente se incurre es que la aplicación no esté montada en memoria en el servidor corporativo. En el cliente aparece un mensaje como el siguiente:

Ilustración 58 Mensaje de error en el cliente al no estar el Web service en la memoria del servidor

La solución a este segundo problema es recargar la aplicación de producción en memoria, con el comando http://localhost:8080/manager/reload?path=/produccion desde un navegador.

Por otra parte, al usar los métodos, no siempre se genera el mensaje 9999-Método exitoso. En ocasiones habrá situaciones especiales que hagan que la transacción falle. He aquí los escenarios más comunes, sus causas y su solución.

Mensaje	**Problema**	**Solución**
5000 - agenciaid: M1189 con password: moreno no son validos. Contacte al administrador	El usuario está mal escrito o no existe en la base de datos MySQL.	• Ir a los catálogos del sistema, corregir el usuario o password y reintentar el método. • Contactar al administrador del Web Service para que se le asigne un nuevo password.
5000 - agenciaid:M1188 con password:moren no son validos. Contacte al administrador	El password está mal escrito o no existe en la base de datos MySQL.	• Ir a los catálogos del sistema, corregir el usuario o password y reintentar el método. • Contactar al administrador del Web Service para que se le asigne un nuevo password.
7001-El XML es inválido. Detalle:the value does not match the regular expression "([0-9]|[a-zA-Z]){17}".	En el Método AltaUnidad se está enviando un número de serie no válido, por ejemplo “NOSON17”	• Corregir el valor erróneo. • Volver a usar el método AltaUnidad.
1217 - agenciaiddestino M1334 no es valida. Contacte al administrador	En el Método Transferencia se está enviando un número de distribuidor no existente	• Contactar con el administrador del sistema para que dé de alta al nuevo distribuidor M1334 • Usar un número de distribudor existente en la base de datos y volver a usar el método Transferencia.
5510 - Error: La unidad 1F3KASL255896SWAH ha sido reportada como vendida por la agencia M1188 (sugerencia: hable a esa agencia para que envien una transaccion CancelaVenta)	Se está tratando de transferir una unidad que ya ha sido reportada como vendida. La venta está asociada con la distribuidora M1188.	• La distribuidora que reportó la venta debe enviar una cancelación con el método CancelaVenta. • La distribuidora que canceló la venta debe transferir la unidad a la agencia que tiene la unidad en su inventario físico.

5210 - No se puede eliminar el vehiculo 1F3KASL255896SWAH del inventario. Ya se ha reportado la venta de este vehiculo	Se está tratando de usar en el numero de serie citado el método BajaUnidad, sin embargo, esa unidad ha sido reportada como vendida y no se puede eliminar del inventario.	• La distribuidora que reportó la venta debe enviar una cancelación con el método CancelaVenta. • La distribuidora que canceló la venta debe dar de baja la unidad con el método BajaUnidad.
1062 - Unidad 1F3KASL255896SWAH ya se encuentra en Inventario	Se está tratando de usar en el número de serie citado en el método AtaUnidad, sin embargo, esa unidad ya existe previamente en el inventario del distribuidor.	• No se requiere acción correctiva. El mensaje 1062 es de carácter informativo.
5500 - Error: La unidad 0123456789AAAAAAA no existe en el inventario. Utilize la transaccion AltaUnidad	Se está tratando de usar el número de serie citado en el método Transferencia, sin embargo, la unidad no existe en el inventario del distribuidor.	• La distribuidora que tenga la unidad en su inventario físico debe agregar la unidad a su inventario con AltaUnidad.
5170 - La venta del vehiculo 1F3KASL255896SWAH ha sido previamente reportada por M1188. Contacte con esta agencia para mas información	Un tercero reportó la venta de una unidad que se encuentra en el inventario físico de otra distribuidora.	• La distribuidora que reportó erróneamente la venta debe enviar una cancelación de reporte de venta con el método CancelaVenta. • La distribuidora que tenga el vehículo en su inventario físico debe reportar la venta.
7001-El XML es inválido. Detalle:the value does not match the regular expression "[a-zA-Z]{3}(.)[0-9]{6}(.){0,10}".	En el Método AltaVendedor, AltaAsesor o AltaCliente se está enviando un valor de RFC inválido.	• Ejemplos de RFCs validos son MOMS760830MX4 y CAB-133589
7001-El XML es inválido. Detalle:the value does not match the regular expression "([0-9]|[a-zA-Z]){5,}".	En alguno de los diecinueve métodos se está enviando una clave de distribuidor o su password con longitud menor a 5 caracteres o con un carácter no valido.	• Corregir la información del catálogo de parámetros correspondiente y reenviar la información.
"7001-El XML es inválido. Detalle:"2004-06-29 00:00:00" does not satisfy the "dateTime" type"	El formato de la fecha es incorrecto, de acuerdo a los tipos de datos definidos en los XML Schemas	• Se debe corregir el formato de fecha enviado a CCYY-MM-DDTHH:MM:SS, en donde la T es el separador entre fecha y hora, CC es siglo, YY es año, MM es mes, DD es dia, HH es hora, MM es minuto y SS es segundo.
7001-El XML es inválido. Detalle:tag name "agencAiaid" is not allowed. Possible tag names are: <agenciaid>	Se está enviando una etiqueta no permitida en la solicitud XML	• Se debe revisar la programación del cliente para asegurarse que los XML de instancia se apegan a los XML Schemas del **Anexo 9**. Pueden usarse como guías los ejemplos de XML SOAP requests propuestos.

7001-El XML es inválido. Detalle:namespace URI of tag "CierraOrden" is wrong. It must be ""	Por definición nuestra, los únicos elementos XML que deben tener definido un namespace es el elemento Envelope y su elemento hijo, Body. Todos los demás nodos deben carecer de namespace	• Se debe revisar la programación del cliente para asegurarse que los XML de instancia se apegan a los XML Schemas del **Anexo 9**. Pueden usarse como guías los ejemplos de XML SOAP requests propuestos.
5320 - El tipodeorden publico- no es valido	Los tipos de orden permitidos son publico, interno y garantia	• Corregir el tipo de orden en el método AltaOrden.
5330 - El tipodeservicio exp_ress no es valido	Los tipos de servicio permitidos son paquete y express	• Corregir el tipo de servicio en el método AltaOrden
5320 - El tipodeoperacion re*pma no es valido	Los tipos de operación permitidos son repma, hyp, mant	• Corregir el tipo de operación en el método AltaOrden
5040 - El cliente:MOMS760830MX4 ya esta dado de alta en la agencia M1188	El cliente con la clave MOMS760830MX4 ya fue agregado previamente a la tabla tblClientes, por lo que la transacción fue rechazada.	• Si se desean corregir datos en el cliente, agregar o quitar teléfonos del cliente, etc, deberá enviarse un método BajaCliente, corregir los datos, y volver a enviar el metodo AltaCliente. • Si al usar el método ListaClientes los datos del cliente son correctos, no se requieren acciones adicionales, salvo recordar la clave para usarla en los métodos ReportaVenta y AltaOrden.
5160 - El tipo de venta vmenude-o es invalido.	Los tipos de venta fijos son flotmenor, varrend, vcomgere, vempleado, vmenudeo, vnarrend, vtagobest, vtagobfed	• Corregir los datos antes de enviarlos con el método de Web services ReportaVenta
5170 - El tipo de pago con-tado es invalido.	Los tipos de pago posibles son arrendamiento, contado y financiamiento	• Corregir los datos antes de enviarlos con el método de Web services ReportaVenta

Tabla 33 Mensajes comunes de error, sus causas y acciones correctivas sugeridas

Debido a que en ocasiones necesitamos rastrear diversos errores en los diseños de XML, generamos una utilería llamada **InstanciaSOAP.java**, cuyo listado aparece en el **Anexo 5**. InstanciaSOAP toma el XML SOAP Request y no lo procesa. Sencillamente lo guarda en el sistema de archivos del servidor, y en lugar de procesarlo y enviar un XML SOAP Response o XML SOAP Fault, envía el mismo XML SOAP Request, con el fin de que el cliente también pueda rastrear, visualizar y depurar los mensajes de XML que envía al servidor, hasta que no tengan más errores y pueda usar apropiadamente **ServicioHTTP.java**. Por tanto, al tener una copia de la solicitud del lado del cliente y otra del lado del servidor, tanto el personal de soporte técnico del corporativo como el personal de la agencia pueden averiguar qué defectos tienen los XML, sea de estructura, nodos, atributos, namespaces, datos, valores, etc.

¿Cómo puede una aplicación de cliente en el lado del distribuidor usar este Web service de espejo? Se usa exactamente de la misma forma en que se usa el Web service normal ServicioHTTP. Para usarlo, se debe abrir el catálogo de parámetros e insertar el URL del servidor como se muestra en la siguiente gráfica:

Ilustración 59 Uso de Web service de espejo para análisis de los XML enviados al servidor

Al usar este servicio de forma local, se puede probar para cualquier método. Por ejemplo, este servicio se uso para el método AltaOrden, con el siguiente resultado de XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ReportaVenta>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>01234567890123456</serieVehiculo>
        <rfc_o_curp_cliente>MOMS760830MX4</rfc_o_curp_cliente>
        <tipoventa>vmenudeo</tipoventa>
        <tipopago>con-tado</tipopago>
        <rfcVendedor>mogt520514kj4</rfcVendedor>
        <km>1000</km>
        <fechaEntrega>2004-07-01T06:27:00</fechaEntrega>
        <term>12</term>
        <tasa>5.6</tasa>
        <pagoMensual>2200</pagoMensual>
        <serieCambio />
        <kmCambio>0</kmCambio>
      </ReportaVenta>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

En este caso, el servicio deja ver que el nodo tipopago tiene el valor erróneo con-tado. De hecho, la utilería fue util para generar los XML del **Anexo 7**, cuyo fin es mostrar los ejemplos estructurales de XML.

5 Resultados

¿Fueron alcanzados los objetivos de la hipótesis? **¿Qué resultado obtuvimos al tratar el problema residente de la recepción de información extemporánea?** Pues bien, en éste caso, los Web services promovieron que tanto los reportes de órdenes de reparación en taller como los reportes de ventas de unidades nuevas se generan "en línea", es decir, ahora las distribuidoras ya generan una transacción al momento de facturar ventas y órdenes, no después o hasta el final de mes. Eso nos dio un mejor control sobre los números de órdenes, al tener en la base de datos los

También alcanzamos el objetivo específico de los Web services orientados a órdenes de reparación en el sentido de que tuvimos un mejor intercambio de información en tiempo real, o, en el momento en que se genera, sobre datos de taller (Aperturas, Cierres, Importes, Cancelaciones, etc), para posteriormente reenviar dicha información a los procesos de solución de quejas y entrevistas del punto de vista del cliente.

Al usar los web services mejoramos el intercambio de información entre el corporativo de la industria automotriz y sus distribuidores, a fin de tener datos en línea y estandarizar las prácticas en la red de los distribuidores. Diseñamos los Web services para que nuestras funciones de negocios estuviesen autocontenidas, es decir, que fueran unidades independientes y que operaran sobre las bases de Internet. Además, sus especificaciones estrictas les permiten trabajar juntos y con otros componentes similares.

Por otra parte, **¿Qué ocurrió con la recaptura recurrente de información en todas las distribuidoras?** La interfaz Java Web services trajo beneficios tanto para los distribuidores como para el corporativo. Por el lado del distribuidor ahora ya se aprovecha la información existente en la distribuidora, lo que elimina la tarea de duplicidad de captura, y los datos tienen una mejor calidad y son útiles tanto para los sistemas locales como para los corporativos. Por el lado del corporativo, al tener un renovado flujo de información impulsado por las interfases estándar se tuvo un mayor volumen de datos de órdenes de reparación, más conciso, con mejores porcentajes de validez, datos indispensable en la elaboración de métricos de objetivos de ventas de unidades y servicio en taller, de tiempo de estancia del vehiculo en agencia, de antigüedad de la unidad en el parque vehicular, inventarios en línea, de lealtad a la marca y lealtad al distribuidor. Todo en una solución.

En tercer lugar, **¿Ya se revisan los datos para determinar que información es de prueba y que información es real?** A pesar de que el distribuidor en ocasiones llega a ser descuidado para capturar sus datos antes de enviarlos, no será más el mismo caso en el punto de recepción, es decir, asumir que los datos son correctos. Los mecanismos de validación básica que programamos al momento de recibir cualquier mensaje, ya son parte medular de los Java Web Services. Si bien el distribuidor desconoce cómo procesan los web services los mensajes XML SOAP Request, sí sabe perfectamente como los valida, a través de los XML Schema. Esta medida redujo a cero el número de mensajes mal formados.

Por otra parte, en el momento que lo desee, el distribuidor puede consultar éstos documentos de normas vía HTTP, usando su navegador tradicional. De hecho, en el lado del cliente eventualmente se podría exigir a los proveedores certificados de software que realicen una validación previa de los XML SOAP Requests que envían, comparándolos con los XML Schemas antes de enviar la información a los XML Web services.

Además de lo anterior, un resultado positivo que tuvimos acerca de los reportes de ventas de unidades nuevas es que al no llegar directamente a las oficinas centrales de Estados Unidos, sino que al ser procesados primeramente en el corporativo nacional, ya estamos asegurándonos de conocer y contabilizar los reportes de ventas mensuales a nivel nacional, lo que de otra manera seguiría ocurriendo "de rebote", es decir, seguir esperando a que el distribuidor envíe a Estados Unidos, y Estados Unidos reporte toda la información rechazada al corporativo nacional, en cuyo caso ya no quedan muchas opciones para solucionar la situación.

Un dato notable en los entregables al corporativo es que estamos brindando la versión 2 del sistema (listado que aparece completamente en el **Anexo 5**) debido a que versiones preeliminares del sistema de Web Services *no incluían la validación del XML SOAP Request con XML Schemas*, lo cual era una de las metas primarias de éste trabajo de investigación. En éste sentido, hicimos lo necesario hasta demostrar que se pueden usar los XML Schemas dentro de los Web services, y que la validación trabaja exactamente igual a la validación que realizan los navegadores "inteligentes".

De hecho, los datos en la base de MySQL nos permitieron, a través de sencillas consultas a la tabla tblVentas, conocer a los distribuidores morosos que consciente o inconscientemente envían sus reportes de ventas mensuales de forma extemporánea. Las sanciones aplicadas a dichos distribuidores fueron desde luego, de carácter monetario y crediticio.

En cuarto lugar, **¿Cumple nuestra interfaz Java Web services con las normas corporativas de seguridad?** Pues bien, durante ésta sustentación de argumentos estuvimos dedicados a dejar claro cómo funcionan los Web services. Como una forma de resumir toda esa serie de explicaciones formales que hemos dado en torno a éste concepto, proponemos el siguiente diagrama de "objetos metálicos en tercera dimensión":

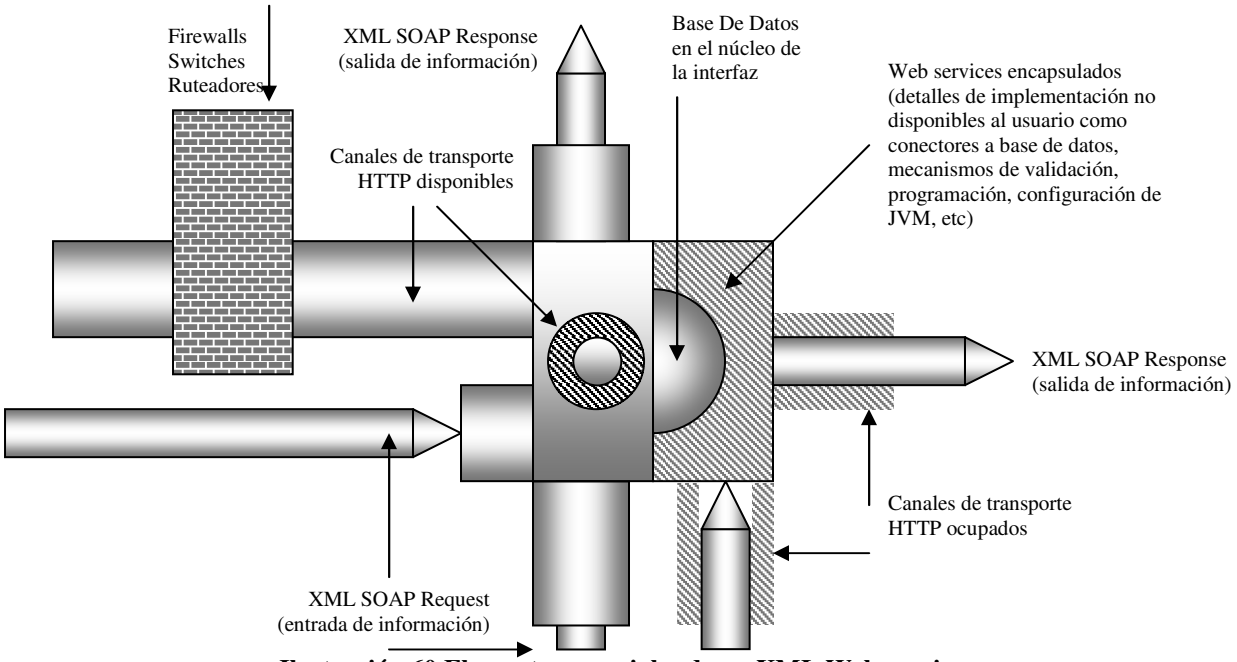


Ilustración 60 Elementos esenciales de un XML Web service

En éste diagrama ilustramos los elementos que componen a un servicio Web. Estos son:

- **El servicio web**. Es el objeto que se representa con un cubo al que se le ha hecho un "corte" para mostrar una de sus "secciones". Se encarga de recibir y validar las solicitudes o "requests", manejar directamente la base de datos y enviar las respuestas o "responses". Los detalles de cómo lo hace permanecen ocultos al usuario.
- **La base de datos en el núcleo de la interfaz**. Es el objeto que se representa con una esfera en el interior del Web service. La base de datos se mantiene aislada del mundo exterior, y nadie puede averiguar qué tipo de base de datos es, cuantos espacios de trabajo tiene, o en qué espacio está trabajando el Web service.
- **Los canales de transporte HTTP**. Son los conductos cilíndricos por los que viajan las solicitudes y las respuestas. Pueden transportar archivos de texto a través de ellos, y como XML es tan parecido a HTML, los canales de transporte HTTP resultan ideales para ésta tarea. Los canales de transporte HTTP traspasan cualquier tipo de firewall, debido a que el puerto 80 de http está por default abierto en todos los firewalls, sean nacionales, extranjeros o en la distribuidora. Todo canal de transporte es auditado por los firewalls. En la figura al tener varios canales de transporte HTTP, se llega a comprender que un Web service realiza transacciones simultáneas, es decir, las procesa concurrentemente.
- **Los XML SOAP Requests**. Están representados con una forma de lápiz que entra al servicio por el canal de HTTP. Son documentos XML con los parámetros necesarios para realizar una transacción en el servicio, sin conocer a detalle qué hace o cómo lo hace.
- **Los XML SOAP Responses**. Están representados con una forma de lápiz que sale del servicio por un canal de HTTP. Son documentos XML con el resultado de una operación solicitada a través de un XML SOAP Request, sea esta exitosa o errónea. La respuesta es síncrona, lo que quiere decir que a cada solicitud corresponde una respuesta y es entregada casi instantáneamente.

- **Los firewalls**. Están representados con una forma de pared de ladrillos. Como todo mundo sabe, los firewalls revisan el contenido de todo lo que pasa por los canales de HTTP, y en términos generales, por los canales de cualquier tipo de protocolo.

De éste diagrama podemos resumir que los Web services son piezas de software compartidas a través de redes, proveyendo servicios y aprovechando protocolos universales. Además se adaptan a las normas corporativas de seguridad al utilizar solamente el protocolo HTTP por el puerto 80.

¿Qué reflejó la investigación al atacar el problema de la ausencia de descripción de los datos? Este problema fue fácilmente abatido desde el momento en que, en lugar de codificar los mensajes en ASCII tradicional, los codificamos en XML SOAP. Comprobamos que XML, en cualquiera de sus gramáticas, es el "ASCII del futuro" porque nos dio un sencillo modo de interpretarlos al no requerir un "layout" por separado. Todo documento XML contiene en sí mismo la estructura y los metadata necesarios para interpretar correctamente los datos. De hecho, para ayudar al distribuidor y al proveedor certificado a utilizar nuestro servicio, ademas de cursos en la red satelital, dimos contratos electrónicos en formato WSDL.

¿Cómo nos fue con la inversión de tiempo y recursos materiales? En los anexos del presente trabajo de investigación colocamos cuidadosamente todos los componentes del producto de software desarrollado para dar solución al problema planteado, y que declaramos como entregables para el corporativo.

Del lado del servidor en el corporativo se tiene:

- El programa en SQL para la definición de estructuras de datos en MySQL (**Anexo 2**)
- Código de JDBC para MySQL en entorno multiproceso o de uso concurrente (**Anexo 3**)
- Código del Sistema de Java Web Services Versión 02 (**Anexo 5**)
- Guía de instalación de la interfaz de Web services (**Anexo 6**)

Del lado del cliente en el distribuidor definimos los entregables:

- Diagrama Entidad Relación del Cliente (**Anexo 12**)
- Código de Creación de la base de datos del cliente (**Anexo 13**)
- Proxies en el Cliente (**Anexo 14**)

Ésta solución consideramos que es versátil, ya que en el lado del corporativo se tiene un servidor de base de datos MySQL y una Java Virtual Machine para Web services. De ambas, se tienen versiones disponibles para los sistemas operativos Windows de Microsoft, Linux, HP UX, Sun Solaris, etc, por lo que situar un hogar para almacenar ésta solución no es difícil, y menos pensando que en el corporativo ya se tenía una computadora que estaría dedicada exclusivamente para producción, aparte de la computadora en la que a prueba y error, desarrollamos la aplicación. No está demas decir que MySQL, Java, y sus Bibliotecas de funciones son de distribución gratuita

En ésta tónica, el cliente tampoco está esclavizado a algún sistema operativo o arquitectura de hardware en particular, De hecho, da igual desarrollar un cliente en Windows de Microsoft, que hacerlo en los otros sistemas operativos supracitados. La única condicional es encontrar la biblioteca adecuada que se adapta al lenguaje de programación seleccionado por el distribuidor, o el proveedor certificado de software en éste caso.

El cliente al igual que el servidor de Web services no es de fabricación complicada, costosa o a largo plazo por causas de alta complejidad. Más bien, el cliente de Web services puede fabricarse hasta con librerías simples como MSSoap Toolkit para MSAccess como lo hicimos en éste trabajo, o bien con la librería de XML SOAP que sea de la preferencia y necesidad del programador de sistemas certificados.

¿Qué tan rápido o que tan lento fueron los Web services adoptados por los distribuidores? He aquí una tabla que muestra el tiempo contra el número de concesionarios con la capacidad de enviar por Web services el año pasado y a principios de éste año:

mes	Num. Distribuidores	Num Distribuidores/Num total de distribuidores
Febrero-2003	0	0.00%
Marzo-2003	3	2.00%
Mayo-2003	11	7.33%
Junio-2003	35	23.33%
Julio-2003	53	35.33%
Agosto-2003	53	35.33%
Septiembre-2003	76	50.67%
Octubre-2003	90	60.00%
Noviembre-2003	130	86.67%
Enero-2004	150	100.00%

Tabla 34 Adopción de la tecnología de los Web services en las agencias distribuidoras

La línea acumulativa tiene la siguiente apariencia:

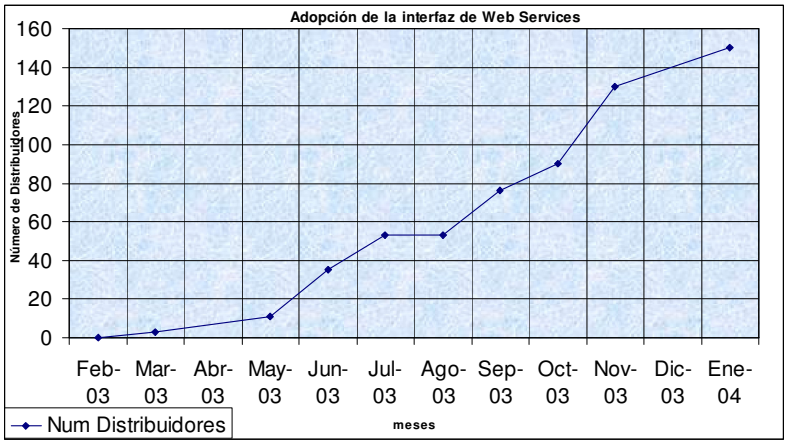


Ilustración 61 Linea acumulativa de adopción de la tecnología

¿Qué pasa si se traza una gráfica con los datos acumulativos de la segunda columna? En tal caso se tiene un diagrama de frecuencias acumulativo absoluto:

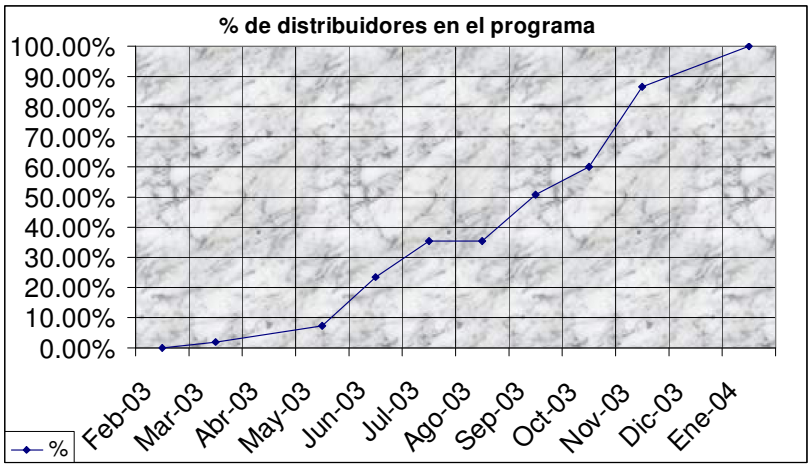


Ilustración 62 Línea acumulativa absoluta de adopción de la tecnología

De ésta última gráfica concluimos que el 80% de los distribuidores adoptaron la nueva metodología en 9 meses y medio, lo que representó un logro satisfactorio si se tiene en cuenta que el distribuidor no estaba familiarizado con el concepto de los Web services.

¿Qué medidas se tomaron para que todos los distribuidores enviaran información a través de los Web services? Se creó un cliente temporal de Web services similar al de **la sección 4.5**, conocido como las "plantillas de captura", que caducó el 31 de marzo de 2004. Este cliente temporal de Web services fue una magnífica ayuda para los distribuidores que aún no contaban con un sistema automotriz certificado, o que sencillamente se encontraban en una migración a finales de Noviembre de 2003.

En septimo lugar, **¿realmente resulto un sistema heterogéneo trasladado en uno homogéneo?** Antes de contestar tal pregunta, proponemos el siguiente diagrama:

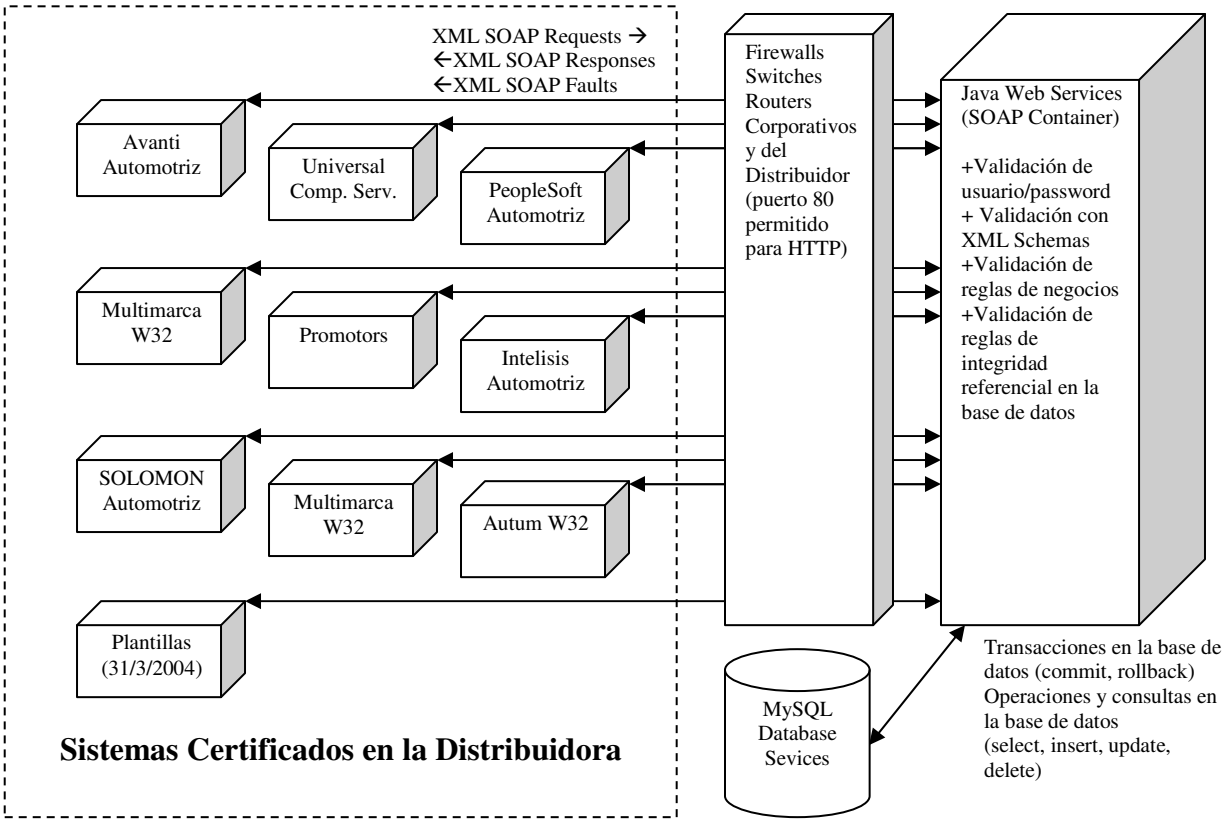


Ilustración 63 Una diversidad de proveedores de software genera un sistema heterogéneo

En el lado izquierdo aparecen todos los sistemas que el corporativo certificó. En el centro se encuentra el firewall, el la parte derecha los Web services y en la parte inferior los servicios de base de datos.

El sistema es heterogéneo porque cada sistema corporativo maneja su perfil técnico, soporte, condiciones comerciales, implantación, lineamientos de autos nuevos, refacciones y servicio, administración de distribuidor y lineamientos corporativos de una forma muy diferente a como lo hace su competidor. Eso nos da al menos 9 diferentes tipos de proveedores de software en un ambiente heterogéneo, de libre competencia.

Los Web services añaden una capa de abstracción homogénea al sistema, porque no importa que tan diferente sea un proveedor de software con respecto a otro, si se apega a los lineamientos de Web services, y se limita a enviar solicitudes de transacción en formato XML SOAP y de la misma forma recibir y procesar esas transacciones, también en formato XML SOAP.

Eso nos permite ver todos los proveedores de software y por ende, a todos los distribuidores, como una sola entidad que se conecta para enviar y recibir sus reportes de ventas y órdenes de reparación al momento de hacer aperturas o facturación.

El sistema heterogéneo se vuelve homogéneo como se muestra en la siguiente figura:

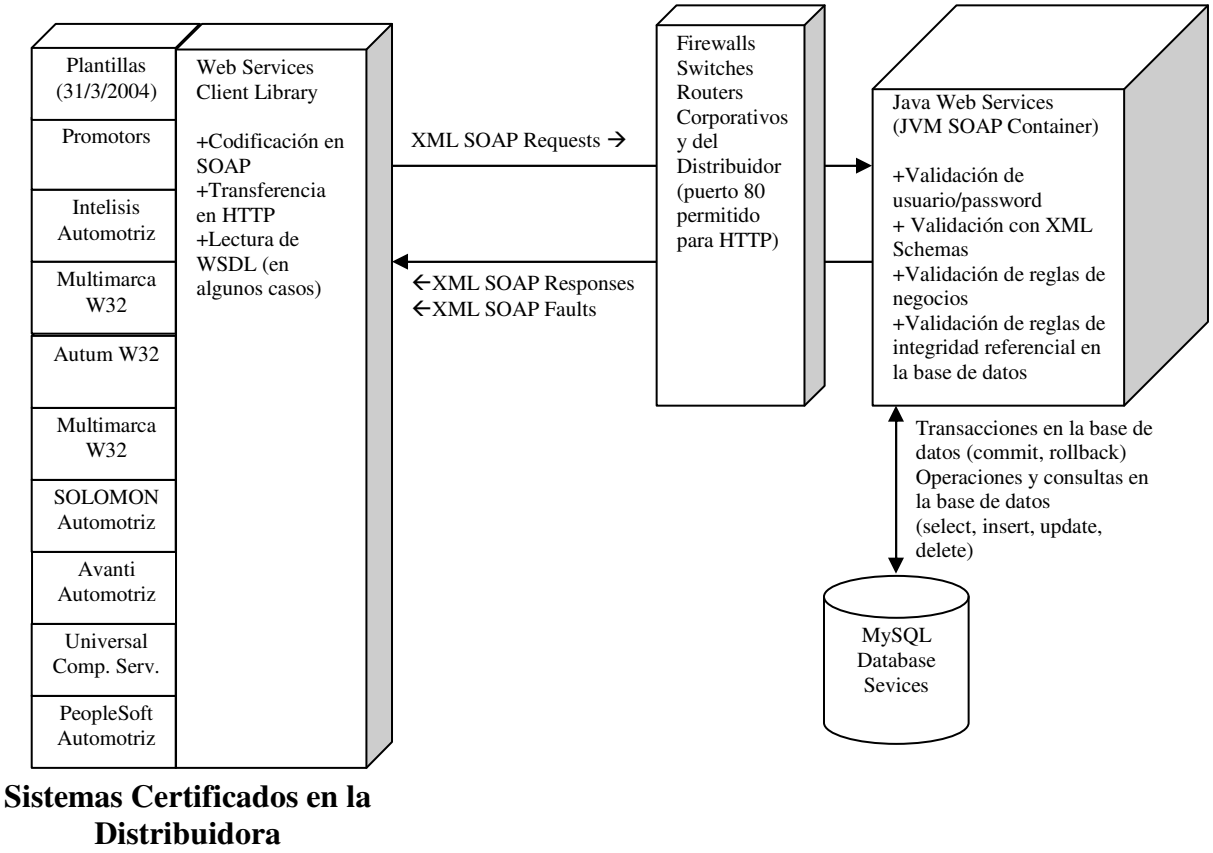


Ilustración 64 Los Web services trasladaron el sistema heterogéneo en uno homogéneo

Lo que decimos con éste último diagrama es que para los Java Web services es transparente e irrelevante a la vez saber qué aplicación le envía datos, siempre y cuando cumpla con las normas de la red de comunicaciones, y los datos sean correctos, y estén perfectamente estructurados en XML SOAP. De tal suerte es tan factible usar los métodos de Web services desde Intelisis, como de SOLOMON. El distribuidor seleccionó el producto que mejor se adapto a sus necesidades, su bolsillo y su infraestructura interna[341].

En nuestro nuevo escenario configuramos una base de datos, y fabricamos un servicio Web para que le diera el aislamiento, la seguridad y la consistencia necesaria, a la par que definimos los estándares de programación de los clientes HTTP-SOAP para la integración de todos los sistemas en la distribuidora. De éstos elementos tenemos obtuvimos que:

a. La base de datos soportó adecuadamente la información que el corporativo desea explotar de diversas maneras, tanto de ventas y servicio, como de inventario y transferencias.
b. Evitamos que el distribuidor volviera a capturar información de órdenes de reparación en taller ni de reportes de ventas de unidades nuevas, si dicha información ya se encuentra en su sistema local. En tal caso, utilizamos la misma información tanto para los sistemas certificados como para los sistemas corporativos.
c. Los archivos de órdenes de reparación en taller los transferimos automáticamente a través de un enlace HTTP sin retrasos en la transferencia de información.

[341] En éste caso no reparamos en citar por nombre a los proveedores de productos de software automotriz en México porque tales atienden a varias firmas como Honda, Toyota, Datsun, BMW, VW, etc.

Fueron importantes los Web services en el sentido de que nos permitieron la interacción de sistemas de compañías con muy diferentes antecedentes, por ejemplo, ligar los sistemas de los distribuidores con los del corporativo de una manera rápida y sencilla. Además, esta arquitectura estimuló a la comunidad a realizar más negocios electrónicamente.

En octavo lugar, **¿Se manejaron cuatro tipos de información en los Web services?** La respuesta es afirmativa con la mención de los métodos de Web services clasificados por tipo de utilización:

- Para **inventarios** se manejaron los métodos *AltaUnidad*, *BajaUnidad* y *AltaInventario*
- Para **transferencias** se implementó el método *Transferencia*
- Para **ventas** se programaron los métodos *AltaVendedor*, *BajaVendedor*, *ListaVendedores*, *AltaCliente*, *BajaCliente*, *ListaClientes*, *ReportaVenta* y *CancelaVenta*
- Para **servicio** se crearon los métodos *AltaAsesor*, *BajaAsesor*, *ListaAsesores*, *AltaCliente*, *BajaCliente*, *ListaClientes*, *AltaOrden*, *mportesOrden*, *CierraOrden* y *CancelaOrden*

Estos métodos permitieron que se manejaran los inventarios reales de los distribuidores en Internet, así como conocer internamente los procesos de transferencia e intercambio de unidades, y llevar un registro de quién los solicita, para las reclamaciones correspondientes.

Por otra parte, **¿La interfaz fue programada utilizando estándares conocidos y tecnología de punta?** Así fue si se recuerda que los Web services fueron programados con Java 2 Standard Edition, en conjunción con Java Web Services Developer's Package, que es un subconjunto de Java 2 Enterprise Edition. De hecho, en el mercado ésta es una buena opción para programar Web services. Existen mas, como las dadas por Microsoft Visual Studio .NET, que es un subconjunto de toda la tecnología .NET, así como soluciones para Oracle, para Sybase y soluciones de otros grandes fabricantes.

Los web services hacen posible la publicación y utilización de todo tipo de información y el negarse a utilizar dicha tecnología es equivalente a darle la espalda al desarrollo tecnológico de la empresa. Todo desarrollo tecnológico resulta en beneficios económicos para cualquier organización, y la industria automotriz no es la excepción en éste caso.

Finalmente, **¿aportó la interfaz un nuevo volumen de datos en las distribuidoras?** ¿Qué ocurrió en las agencias distribuidoras al haberse adoptado la arquitectura de los Web services? En el **Anexo 15** muestro información pertinente tocante a las órdenes de reparación de 4 distribuidoras, en donde muestro cómo reaccionó ésta variable crítica del número de órdenes de reparación con la introducción de los Web services. Ésta muestra de información la tuve disponible hasta Noviembre de 2003, mientras todavía me encontraba laborando en el corporativo de la industria automotriz ubicado al oeste de la ciudad de México.

Debido a mi determinación de no utilizar nombres reales de las razones sociales de los distribuidores automotrices, me limité a etiquetarlos con el nombre de la calle de su ubicación geográfica, como "Avenida Universidad", "Calzada de la Viga", "Tlalpan" y "Carretera Mexico Toluca". Entre los factores en común que se encuentran en estos casos particulares está el hecho de que:

- La línea de cuota de órdenes tiende a ser un límite deseable para los datos reales.
- Antes de utilizar los Web services había una baja probabilidad de alcanzar la meta de órdenes de reparación establecida por la línea de cuota.
- Después de utilizar los Web services hay una mejor probabilidad de alcanzar y superar la cuota. En los cuatro casos, las órdenes reportadas tienden a quedan muy por encima de la línea de cuota.
- Después de utilizar los Web services, tiende a estabiizarse el comportamiento en el envío de la información, y cuando no se alcanza la cuota, es porque se generó una variación especial en algún mes de la agencia distribuidora.
- El número de órdenes enviadas tiende a ser igual al número de órdenes correctas, sin datos inválidos.

Caso I. En la distribuidora de Avenida Universidad, al sur de la ciudad de México, encontramos que durante tres meses consecutivos –de enero a marzo de 2003- no se alcanzó el objetivo de recepción de órdenes de reparación. En los siguientes dos meses, abril 2003 y mayo 2003, observamos que los objetivos se alcanzaron debido a que se redujo el número de órdenes requeridas por el corporativo, no por algún cambio sustancial en el sistema. En éste caso se observó una mejoría a

partir de la introducción de los Web services, ya que el porcentaje de órdenes correctas mejoró de 93.17% a 94.07%, en promedio, y el promedio de porcentaje de objetivos se elevó desde un 77.64% hasta un 174.96%. Adicionalmente, durante cinco meses consecutivos se alcanzaron y sobrepasaron los objetivos de órdenes de reparación cerradas. Lo que las cifras arrojaron fue también una tendencia a reducir el número de órdenes reparadas por mes. En este caso, al haber descartado el método de transmisión, bien se pudieran investigar las causas de ésta desestabilización de información.

Caso II. Las gráficas 3 y 4 hablan acerca de la información recibida de la agencia ubicada en Calzada de la Viga al corporativo. Al no haber introducido aún los web services, no se observaba una tendencia a alcanzar los objetivos. De hecho, en los dos datos que se tomaron, enero y febrero de 2003, no se alcanzaron los objetivos de ordenes cerradas debido a tener un pequeño volumen de información y porque la información recibida resultó incorrecta.

Lo que ocurrió en los subsecuentes meses, de marzo a octubre de 2003 resulta interesante, ya que en primer lugar, se dio un cambio en los objetivos de ventas por estar la cuota muy alta en relación a otras distribuidoras. En mayo se generó una baja de órdenes recibidas, y al investigar porqué, se encontró que existieron causas diferentes al modelo de transmisión de los Web services. Después, de Junio a Octubre de 2003, se sobrepasaron los objetivos de ventas de forma consecutiva por 4 meses. Resta decir que el porcentaje de correctas bajó de un 75 a 82.86%, y que el porcentaje de satisfacción de objetivos promedio subió desde un 71.75% a un 109.14%

Caso III. En los datos obtenidos de la agencia distribuidora en calzada de Tlalpan se encontró que en febrero y marzo de 2003, de manera consecutiva por dos meses consecutivos no se tubi una buena aproximación al objetivo de órdenes. De hecho, en los siguientes tres meses, de abril a junio de 2003 se alcanzaron los objetivos de órdenes no por algún cambio sustancial en sus métodos de envío o procesamiento, sino porque el corporativo redujo el parámetro de cuota de 529 órdenes a tan solo 270.

Satisfactoriamente, al introducir un sistema certificado con su interfaz de Web services, durante 4 meses consecutivos se sobrepasaron los objetivos de órdenes de reparación cerradas. En ésta curva se tiene una tendencia de disminución de órdenes procesadas, cosa que es completamente ajena al sistema de Web services.

En resumen, pese a que el promedio del porcentaje de órdenes correctas bajó de un 99.82%, el promedio del porcentaje de objetivos subió desde un 118.66 hasta un 194.39%

Caso IV. En la agencia ubicada a un costado de la carretera México-Toluca también se tienen resultados halagüeños. Antes de introducir los sistemas de interfases en línea, por tres meses consecutivos no fueron alcanzados los objetivos de ordenes abiertas. Durante los siguientes dos meses, abril y mayo de 2003, apenas se alcanzaron los objetivos no por algún cambio sensible en el sistema, sino por una reducción en la cuota de 368 a solo 188 órdenes..

Con la introducción de los Web services, el panorama también mejoró para ésta agencia distribuidora, debido a que por cinco meses consecutivos se sobrepasó la cuota de 188 órdenes. Pese a que el % de órdenes abiertas bajó un poco, de 96.04 a 87.65% en promedio, el % de objetivos fue superado de un 78.81 a un 190.21%, lo que produce una recuperación de datos considerable. Resta decir que en la **gráfica 8** de éste **Anexo 15** se observa una tendencia de reducción moderada, cuyas causas deben inhibirse para mantener el proceso controlado.

Por otra parte, el **Anexo 15** también arroja información sobre ventas de unidades nuevas que hace pensar en la utilidad de los sistemas certificados y las interfases de Web services al complementarlos. Las **gráficas 9 a 13**, a diferencia de las **gráficas 1 a 8**, analizan los datos de toda la red de distribuidores, no de solamente distribuidoras aisladas. De hecho, existen tres tipos de unidades nuevas, que son automóviles, camionetas mixtas y camiones de uso rudo.

En **la gráfica 9** se tienen los objetivos de ventas de automóviles contra sus ventas a menudeo reales. Los primeros seis puntos corresponden a los primeros seis meses de ventas en los que no estaban implantados los sistemas usuarios de la interfaz Java Web services. Ahí, se observa una baja tendencia a aproximarse a la curva de objetivos estando las ventas de menudeo muy por debajo de la cuota. No obstante, en los siguientes 5 meses se tiene una mejor aproximación a la curva ideal. Inesperadamente, se tiene una depresión en los meses de julio y agosto de 2003 al venderse 4749 y 4667 unidades, respectivamente. Ésta variación se debió a que se tuvo una fuerte competencia por el mercado con respecto a otras firmas automotrices que lanzaron nuevos modelos ése año. A pesar de ello, en los siguientes 5 meses se obtuvo la tendencia a alcanzar los objetivos al ser las curvas de objetivos y menudeo casi paralelas.

Similarmente, la **gráfica 10** aporta información de ventas de camionetas. En los primeros cinco meses se observa la dificultad para tocar los objetivos de vntas, en los siguientes seis meses, para sopresa de los directivos, se sobrepasaron consecutivamente los objetivos de ventas, de febrero a junio de 2003. En julio, la competencia del mercado y otras causas también ocasionaron que se dieran variaciones que produjeron una oscilación en la curva de la cuota hasta enero de 2004. En los meses que no se alcanzaron los objetivos, obviamente fueron por causas ajenas al sistema de información.

La **gráfica 11** aporta detalles sobre las ventas de camiones antes y después de la introducción de los Web services. Aquí, los primeros 5 puntos representaron los primeros 5 meses en los que no se alcanzaron de manera razonable los objetivos de ventas. Después en los siguientes meses se observa que la curva de venta a menudeo del distribuidor al cliente final se apega mejor a la curva de objetivos. En los camiones también se dio la afectación de la variación de mercado que existió en julio, agosto y septiembre de 2003, al venderse solamente 2203, 2025, y 2837 unidades ésos meses.

Al sumar las cifras de objetivos y unidades vendidas por mes y generar las curvas correspondientes, proveemos la **gráfica 12**, que resume que, en los primeros seis meses, se tuvieron serios problemas para alcanzar los objetivos de ventas, y que en los siguientes meses, de febrero de 2003 a enero de 2004, la curva de ventas tuvo un desempeño más faborable, al poseer un mejor porcentaje de satisfacción mes con mes. En éste caso, no es tan sencillo sobrepasar los objetivos de ventas como el de las órdenes de reparación, ya que se requiere un fuerte trabajo de mercadeo, campañas, promociones y eventos especiales, que llegan a ocasionar cuantiosas inversiones a la compañía.

En la **gráfica 12** también se muestra el efecto de la variación de mercado de julio y agosto de 2003, al vender de manera crítica solamente 8400 y 7735 unidades, respectivamente. Días negros para el corporativo y en términos generales, la industria automotriz.

¿Qué es lo satisfactorio para la industria del transporte? No hay norma escrita para esto, pero en sistemas siempre tomábamos el parámetro de 80% de ventas reales/cuotas, Así, en los primeros seis meses, de agosto de 2002 a enero de 2003, se tuvo una proporción de satisfacción 1/6, proporción muy baja si se observa que se tiene un mejor porcentaje de satisfacción de febrero de 2003 a enero de 2004, en donde se tiene una razón de 9/12, siendo julio de 2003, agosto de 2003 y enero de 2004 los peores meses de ventas. Se invita al lector a revisar la **gráfica 13** del **Anexo 15** para que compruebe que se tiene una mejor probabilidad de alcanzar los porcentajes de niveles de satisfacción con la introducción de los Web services.

Si bien estas cifras no son un análisis estadístico formal, éste análisis numérico para ventas y órdenes de reparación es suficiente para probar que los Web services incrementan el volumen de datos recibido por el corporativo.

6 Discusión de Resultados

¿Fueron los resultados satisfactores de los obstáculos en la transferencia de repotes de ventas y ordenes de reparación? A continuación esta tabla muestra los problemas existentes y la forma en que la Interfaz Java Web services dio solución a cada uno de éstos rasgos:

#	Problema	Solución
1	**Información extemporánea**. Tanto los reportes de órdenes de reparación en taller como los reportes de ventas de unidades nuevas llegaban tardíamente en el mejor de los casos fuera de tiempo en el peor de los casos.	**La interfaz hizo que los sistemas certificados puedan enviar la información "en línea"**. Cuando se factura una unidad, se ejecutan los métodos AltaUnidad, y ReportaVenta correspondientes. Cuando se factura una órden de reparación se lanzan los métodos AltaOrden y CierraOrden, según sea el caso. Con ello se evita envíar la información extemporáneamente.
2	**Recaptura de órdenes de reparación en taller y reportes de ventas de unidades nuevas**. El distribuidor, después de capturar éste tipo de información en sus sistemas propios, después debía hacerlo en los corporativos, sin tener un medio común para compartir la información. De hecho, ése era el pretexto de enviar tardíamente la información.	**La interfaz evita la recaptura de información**. El distribuidor no tiene que capturar la misma información dos veces porque la interfaz integra la información del sistema certificado en el sistema corporativo, debido a que se trata de la misma información. Los XML Web services hacen posible esto.
3	**Información inválida**. La información se recibía en el corporativo sin la validación básica por deficiencias en los sistemas del distribuidor Por otra parte, en las agencias distribuidoras de automóviles, se tiene alta rotación de personal en ventas, servicio, y todas las áreas. Tales eventos conllevan el problema persistente de que el personal recién contratado tiende a enviar datos inválidos debido a su inexperiencia en el procesamiento y transferencia de información. El no establecer algún punto de revisión de datos de los reportes de ventas y servicio ocasionaba una detección tardía de errores. Tanto los proveedores de encuestas como la sede en Estados Unidos, rechazaban datos inválidos de los distribuidores, siendo que el corporativo podía evitar tal problema. Alto índice de información errónea por distribuidor. Este hecho desemboca en que el distribuidor no alcanza sus objetvos de ventas u ordenes porque al cliente le faltaron teléfonos, porque el numero de serie de la unidad venía mal escruto, porque la dirección del cliente era errónea, y muchas otros detalles evitables.	**La interfaz contiene métodos de validación de información recibida**. La interfaz introduce la tecnología de validación basada en XML Schemas. Este sistema de validación, al estar situado en la fase de recepción de la información, evita la recepción de información claramente inválida. La interfaz verifica los tipos de datos que toma de los proveedores certificados y entrega al corporativo de la industria automotriz usando un mecanismo flexible y adaptable a los bruscos cambios de las políticas de la organización. En caso de que la interfaz detecte un conjunto de datos no válidos en algún mensaje recibido, desecha dicho mensaje y responde al distribuidor la causa por la cual desechó el mensaje.
4	**Brecha de seguridad en los firewalls.** Los reportes de ventas de unidades nuevas eran capturados usando una terminal de emulación VT3270 hacia un equipo AS400, hecho ue abría puertos no permitidos en los firewalls nacionales y norteamericanos, ocasionando un descontento para los administradores de dichos equipos por ser una amenaza potencial. Los archivos de órdenes de reparación en taller eran **transferidos** a través de un cliente y un servidor de ftp, lo que también abre una brecha en los firewalls por los puertos 20 y 21.	**La interfaz cumple con las normas de seguridad corporativas**. Esto lo hace desde el momento que utiliza un protocolo de transporte seguro, basado en texto, como HTTP. La interfaz esta situada en una red corporativa, con lineamientos preestablecidos de seguridad y restricciones en transferencia de datos (switches, routers, firewalls, telephony earth stations, personal earth stations, hybrid earth stations, equipo satelital, etc).

5	**Ausencia de descripción en los datos**.Para las órdenes de reparación, la información transferida se encontraba en formato plano o "flat ASCII". Interpretar tales archivos requeria un "layout" por separado, cosa que era riesgosa porque si el layout cambiaba al agregarse o eliminarse longitudes, encabezados, pies de página, etc, el archivo ASCII no daba señales visuales de ello.	**La interfaz utiliza datos autodescritos o metadatos**. Los datos están perfectamente autodescritos al usar XML para codificarlos, preferentemente usando la gramática particular SOAP.
6	**Infraestructura de soporte de costo elevado** Las órdenes de reparación se recibían en un servidor FTP del corporativo. Los primitivos servidores de éste protocolo, como las Digital MicroVax 3300 con Multinet, tenían la ventaja de ser contactados también a través de la vía telefónica con el protocolo BLAST, pero debido a que el tiempo vuelve obsoleta a cualquier computadora, éste servidor adquirió la característica de ser de caros mantenimientos preventivos y correctivos.	**La interfaz se manufacturó con costos de programación mínimos**. MySQL, Java y sus Bibliotecas, y Microsoft SOAP Toolkit, son de distribución gratuita, lo que no acarreó un costo adicional a la compañía. La interfaz se programó en menos de un año y su adaptación por parte de la red de distribuidores fue en nueve meses y medio, lo que no generó grandes pérdidas de tiempo o recursos.
7	**Diversidad de infraestructuras en los distribuidores**. Los sistemas operativos del distribuidor en ocasiones no eran compatibles con los sistemas para transferir información por ftp, ni para usar la terminal de emulación VT3270. La diversidad de proveedores de software provoca que se tenga un sistema heterogéneo.	**La interfaz trasladó el sistema heterogéneo en un sistema homogéneo**. En vista de que los proveedores certificados de software no desarrollan ni en el mismo sistema operativo ni en el mismo lenguaje de programación, ni con la misma metodología, cada proveedor certificado de servicios se convierte en una variable diferente. Por tanto debe pensarse en una forma de transferencia de información a la compañía que no dependa de dichas variables.
8	Si bien se conocía parcialmente el manejo de reportes de ventas y órdenes de reparación, se **desconocía el manejo de información** entre distribuidores tal como: No se estaba manejando el Inventario de unidades nuevas de distribuidor, información necesaria para publicarse en internet **Ni las transferencias de vehiculos, ni los intercambios** de una distribuidora a otra se estaban registrando en alguna parte.	**La interfaz manejó cuatro tipos de información**. Adicionalmente a los reportes de ventas y órdenes de reparación, la interfaz Java Web services proveyó metodos para manejar información de inventario de distribuidores, transferencias e intercambios de automóviles entre ellos.
9	**Se estaban desaprovechando los protocolos de transporte conocidos HTTP, y nuevos protocolos de transferencia como XML. Además se estaban desperdiciando las nuevas tecnologías de información y los modelos de objetos distribuidos.**	**La interfaz fue programada utilizando estándares y tecnología de punta**. Los Web services están siendo adoptados rápidamente como el estandar en arquitecturas orientadas a servicio, lo que hace que todos los fabricantes especializen sus herramientas para ésta tarea.
10	**Se tenía un bajo volumen de datos recibidos por distribuidor.** Este hecho también provoca que el distribuidor no alcance sus objetvos de ventas u órdenes por ausencia de información.	**La interfaz aportó un nuevo volumen de datos recibido por el distribuidor.** La interfaz estimuló nuevas prácticas de negocios electrónicos en la red, para tener los datos en línea, y recibirlos al momento de que se generen, y no tiempo después.

Éste cuadro muestra de forma muy condensada la situación problemática que existía (comentada en el **capítulo 1)** y los satisfactores que aportó la solución en Web services. Estos diez puntos dan a saber que se eligieron las mejores opciones en tecnología y se tomaron las mejores acciones para dar una respuesta satisfactoria a cada una de las necesidades técnicas y de negocios del corporativo automotriz.

La recomendación para toda industria automotriz es que haga un continuo seguimiento de sus variables críticas de órdenes y de ventas usando herramientas matemáticas tales como el control estadístico del proceso, para asegurarse que todos los distribuidores estén enviando continuamente y sobre su cuota. Tal como un doctor realiza electrocardiogramas periódicos en un paciente, podríamos realizar diagramas de control estadístico de manera continua para conocer los instantes exactos en los que ocurrieron las variaciones especiales del proceso. Una herramienta auxiliar para éste procedimiento podría ser Minitab para Windows. Es importante detectar un Julio y un Agosto de 2003 para darnos cuenta de quién nos está quitando un sector del mercado o quién esta dejando de hacer su trabajo cotidiano.

Otra herramienta que podría introducirse en el estudio de los datos es el "Data Mining" a través de un "Datawarehouse" montado silenciosa y transparentemente en la base de datos, para realizar reportes ejecutivos de la información de manera continua, con la poderosa herramienta del "Datawarehouse" que es el "Datadrill" para estudiar los datos, disponible en los paquetes destinados para ello, como es Business Objects.

Por otra parte, al ser los Web services una solución razonable para crear interfaces en los sistemas automotrices, se puede decir con certeza que ésta metodología es recomendable para proveer interfases estándar a otro tipo de problemas dentro de la industria automotriz, por ejemplo, los relacionados con los pedidos de refacciones, avisos de pago, vehiculos seminuevos, estados financieros y cualquier otro sistema imaginable.

Los Web services pueden involucrarse en cualquier proceso de información, no solo en aplicaciones específicas de la industria automotriz, sino también en el giro de la medicina, gubernamental, del comercio, de comunicaciones, y en donde se demande transferir piezas de información de un lugar a otro, así como verificarlas en el lugar de recepción. Invitamos al lector a que descubra por si mismo la sencillez, flexibilidad y ventaja de crear y utilizar Web services. Además, el concepto de Web services es el mismo, sin importar si se está trabajando en Microsoft.NET, en J2EE o en Apache de GNU, por lo que es conveniente manejar esas opciones a nuestro alcance.

Conclusiones

1. Síntesis por capítulo

En el capítulo 1 expuse el requerimiento del corporativo al departamento en el que laboré. Dicho requerimiento generó una situación problemática, la cual tuve que situar dentro de un marco teórico-conceptual para aislar el problema, identificar sus elementos y plantear posibles soluciones. En éste capítulo también hablé acerca de la primera variable crítica que debe conocer el corporativo a través de los datos de ventas de menudeo y las reglas de negocios en torno a esa información. Definí el número de órdenes de reparación como una segunda variable crítica del negocio y también hablé acerca del tipo de información que debe estar contenida en cada orden de reparación. Una vez más definí reglas de negocios para éste tipo de reporte.

En la parte final del capítulo 1 explicamos los objetivos de la información de los reportes de ventas y órdenes de reparación. Expusimos que estos datos tienen importancia dentro de la compañía, debido a que a partir de esos datos se generan productos diarios de información para otros departamentos dentro de la misma compañía e incluso para proveedores de encuestas en el exterior. Además planteamos la situación problemática de manera gráfica e identificamos las áreas de oportunidad en las que debíamos trabajar.

El capítulo 2 estuvo dividido en seis secciones. En la sección 2.1 respondemos a la pregunta de porqué se usó Java para programar la aplicación. En la sección 2.2 declaramos muchas razones por las que se eligió MySQL como motor de base de datos en el corporativo. En la sección 2.3 identificamos el caso de estudio como un problema de componentes distribuidos, por lo que mostramos el resultado de nuestra investigación en torno a las características de las arquitecturas de los sistemas distribuidos y dimos una breve historia de ellos.

Dentro de la historia del cómputo distribuido encontramos que para compartir procesos –funciones, subrutinas, etc- hay barreras como la diversidad de microprocesadores, sistemas operativos y compiladores, así como una basta diversidad en las arquitecturas de redes. Las soluciones primitivas al cómputo distribuido fueron unas versiones preeliminares de RPC, pero desafortunadamente las versiones originales de llamadas a procedimientos remotos todavía tenían limitaciones de lenguaje, aunque el concepto de RPC como tal llegó a ser rescatable.

En la sección 2.3 respondimos a la interrogante que cuestionaba la existencia de los sistemas heterogéneos. De hecho, dimos una lista de razones por las que alguien desearía desarrollar una aplicación distribuida. De esas razones concluimos que la heterogeneidad de los sistemas es una realidad de nuestros días, y explicamos qué problemática conlleva tal fenómeno en las operaciones de negocios. De hecho, en ésta sección 2.3 dimos a conocer a la comunidad de los ingenieros en computación que todos debemos estar interesados en tratar con los sistemas distribuidos, porque la heterogeneidad de los sistemas es una realidad, la mayoría de las veces inevitable, la cual debemos no solo conocer, sino manejar a nivel tecnológico. No la debemos ver como un obstáculo, sino como un mercado de trabajo que se abre ante nosotros.

En vista de que los ingenieros en computación debemos programar sistemas distribuidos, requerimos de modelos independientes de plataforma así como abstracciones para resolver varios tipos de problemas. De hecho, requerimos ocultar la complejidad de bajo nivel en la medida de lo posible.

Concluimos la sección 2.3 definiendo los criterios para determinar si una aplicación debe desarrollarse en componentes distribuidos o no. De hecho, comparamos tales criterios contra el caso de estudio, y de esta forma justificamos el hecho de que requerimos utilizar una arquitectura distribuida para resolver la situación problemática del caso de estudio.

En la sección 2.4 estudié las alternativas en arquitecturas de componentes distribuidos y determiné que los Web services son la mejor opción para mi caso de estudio. En la sección 2.4.1 me dediqué a describir la arquitectura de CORBA, y mostré las razones por las que CORBA es una buena opción para diseñar y escribir sistemas. Aquí, mostré las definiciones de la especificación de CORBA. Cité los lenguajes de programación que

trabajan con esta arquitectura, y definí porqué son necesarios los mapeos de IDL hacia los lenguajes de programación.

En la sección 2.4.1 definimos el modelo de objetos y el de referencias, y mostramos como estandariza CORBA sus referencias. Además trazamos un esquema general del funcionamiento de objetos distribuidos de CORBA. Así, a partir de ese esquema resumido generamos otro diagrama con la descripción formal de la arquitectura CORBA. Del diagrama concluimos que un componente puede, en un lapso de tiempo, jugar el papel de cliente, y en otro lapso de tiempo, el papel de servidor, dependiendo del contexto que provea el mensaje enviado de un componente a otro. También observamos que las requisiciones a objetos distribuidos son síncronas, síncronas diferidas, de una vía y asíncronas, a partir de la versión 3.0 de CORBA.

En la sección 2.4.1 también situamos la terminología única asociada con CORBA. Sin ésta, no hubiéramos podido tener un entendimiento razonable de la arquitectura. Además aprendimos qué funciones realiza el ORB cuando el cliente invoca una operación a través de una referencia. Las solicitudes también las clasificamos de acuerdo a la forma en que se construyen, estática o dinámicamente. Encontramos que desarrollar en CORBA nos provee transparencia de ubicación del servidor, independencia del lenguaje de programación, de la arquitectura de red y del sistema operativo, así como de protocolo, y en algunos ORB, independencia en el método de transporte.

Los CORBA Requests preferentemente son elaborados con la aproximación de invocaciones estáticas a través de PROXYs CORBA. Aquí definimos que un PROXY provee al cliente la interfaz del objeto destino –que es diferente al concepto de PROXY en el ámbito de switches, firewalls y routers de las redes físicas de computadoras-. Los PROXYs de CORBA delegan la invocación de las operaciones hechas por el cliente al servidor. Además mostramos cómo se construye un PROXY de CORBA. Por otra parte, solamente mencionamos someramente los requests dinámicos en CORBA.

En CORBA las Object Iinterfaces son definidas en el OMG Interface Definition Language. Encontramos que IDL permite que aplicaciones implementadas en diferentes lenguajes de programación puedan interoperar, y que IDL es el mecanismo fundamental de abstracción para separar las interfases del objeto de sus implementaciones. De hecho la “B” de CORBA recuerda al desarrollador que la interfaz está separada de su implementación, o “Broken” en inglés. Por tanto, analizamos qué especificaciones reune IDL, y advertimos que tal lenguaje no es usado para escribir detalles de implementación en los programas. De hecho, IDL no define ciclos for, do, while, o sentencias de control condicional, porque sencillamente IDL no puede ser compilado o interpretado para que se convierta en un programa ejecutable. Concluimos que IDL está diseñado únicamente para declarar interfases de objetos y establecer qué tipo de datos utilizarán esas interfases para comunicarse con tales objetos. En vista de que IDL es un lenguaje puramente declarativo, las definiciones de IDL se enfocan en las interfases de objeto, las operaciones soportadas por esas interfases y las excepciones que pueden ser activadas por dichas operaciones.

Una observación importante dentro de esta sección es que el IDL compiler no es el mismo para todos los lenguajes de programación, es decir, los nombres de los archivos fuente y objeto varían de un IDL compiler a otro.

Por otra parte, el Object Adapter es el pegamento entre los programas sirvientes y el ORB. El Object Adapter complementa al ORB. De hecho, sin los Object Adapters, los ORB tendrían una implementación muy compleja. En éste sentido, los ORBs implementan la independencia del lenguaje de programación para la petición, ya que son servicios distribuidos que manejan ciertas categorías de interfases, y éstas componen a su vez marcos de trabajo para objetos distribuidos. Los servicios que soportan a CORBA controlan el ciclo de vida de los objetos, los nombres simbólicos de dichos objetos, sus eventos, relaciones, publicación, transacciones, acceso serial, propiedades, negociación y consultas. Los protocolos de transporte de CORBA son GIOP e IIOP. También dimos a conocer seis pasos para construir una aplicación CORBA.

Finalmente, describí un entorno de desarrollo CORBA, utilizando uno y dos lenguajes de programación. Aquí me di cuenta de que, en el peor de los casos, la única liga entre los desarrolladores del cliente y el servidor es la definición de IDL utilizan. Además mencioné que los sistemas CORBA son escalables para mejorar su

rendimiento. Al finalizar el capítulo di un ejemplo práctico de un programa en el ORB de Java, e ilustré los pasos específicos que deben seguirse y las herramientas apropiadas que deben usarse para ejecutar una aplicación distribuida en CORBA.

En la sección 2.4.2 hablé de otro protocolo de objetos distribuidos propiedad de Sun Microsystems, llamado Java Remote Method Invocation (de hecho, dentro del lenguaje Java, los conceptos de RMI y CORBA están mezclados en las interfases RMI/IIOP) RMI ha sido diseñado para crear sistemas de objetos distribuidos y una restricción es que todos los elementos deben compilarse como clases de Java. RMI tiene la capacidad para pasar objetos de Java por valor y descargar nuevas clases de Java entre clientes y servidores. RMI es una arquitectura más ligera que CORBA y es parecida a ésta, en el sentido de que maneja clientes, stubs, skeletons y servidores. También cité los paquetes de Java para dar tratamiento a aplicaciones RMI. De la misma forma, mencioné las herramientas de apoyo para hacer aplicaciones y promover la infraestructura RMI en tiempo de ejecución.

Así como en CORBA, antes que un servidor RMI haga sus servicios disponibles, debe ser configurada una infraestructura que funcionará en tiempo de ejecución. Por tanto citamos los pasos que componen al proceso para desarrollar una aplicación utilizando los lineamientos de la arquitectura RMI. RMI utiliza el protocolo de comunicación JRMP y de éste puede decirse que, así como CORBA puede envolver GIOP con HTTP (IIOP), JRMP también puede venir envuelto en el protocolo HTTP.

Ésta sección concluye con la reflexión que establece que es menester pensar en CORBA para Java cuando alguien nos obligue a usar un archivo IDL para hacer mapeo de clientes en servidores distribuidos (esto es, de interfaz a código), y pensar en RMI/IIOP cuando se tienen servidores Java RMI que necesitan exponerse en términos de interfases IDL. Además, un ejemplo práctico mostró cómo desarrollar un programa de forma distribuida en RMI.

En la sección 2.4.3 hablamos sobre la arquitectura de Web services, que es un abanico de términos que describe a una colección de protocolos estándar existentes de la industria, así como servicios utilizados para facilitar una línea de nivel base para la interoperabilidad entre aplicaciones. Mencionamos que dichos estándares son SOAP, WSDL, XML Schemas y UDDI. Los Web services están construidos en bloques de XML y son importantes porque estos simplifican ampliamente la creación e integración de sistemas distribuidos a gran escala.

Aunque HTTP y HTML hizo posible para los consumidores utilizar páginas Web remotas desde hace mucho tiempo, lo que faltaba era simplificar la integración de los sistemas de negocios y tener una forma de compartir datos y servicios de software a través de Internet. Por eso surgió la necesidad de transmitir XML sobre HTTP. De hecho, los Web services conjugan los mundos de la programación e Internet, que estaban aislados entre sí, usando un contenedor SOAP.

Los contenedores SOAP convierten los mensajes XML en llamadas a procedimientos locales en una computadora. Un XML puede ser enviado de una computadora a otra por cualquier medio, pero HTTP es la capa de transporte más común. Para obtener la dirección y la información del mensaje, los clientes SOAP leen un archivo WSDL que describe al servicio Web.

Así como sucedía en CORBA, las implementaciones de PROXY ocultan los detalles de implementación SOAP. Una solicitud SOAP es aceptada por un servlet, quien revisa el campo SOAPAction. Así, el contenedor usa el URI del POST para buscar el Web service objetivo, traducir la carga de datos de XML e invocar el método en el componente.

La sección 2.4.3 también contiene los elementos básicos de un servicio web de ejemplo, y definió tres fases de implementación. Otra forma en que analizó los Web services fue estudiando los bloques de construcción de Web services por separado, a saber, Discovery, Description, Message Format, Encoding y Transport.

Es vital el que al día de hoy el Ingeniero en Computación esté familiarizado con los Web services, debido a que las aplicaciones del futuro serán ensambladas a partir de un conjunto de servicios diseñados para ejecutarse en

red. Los Web services, por ésa razón, cuentan con aceptación unánime. La convergencia de factores tales como HTTP, WSDL, SOAP, XML, y UDDI posibilitan el mismo tipo de efecto en red que permitió que HTML, llegara a ser rápidamente el estándar a la mano para compartir información en la red. Las tecnologías de cómputo distribuido previas tales como CORBA, DCOM y RMI nunca han tenido tal aceptación unánime.

Desarrollar en Web services es económicamente conveniente debido a que se pueden construir aplicaciones distribuidas de buena calidad en una fracción del costo de las interfases distribuidas tradicionales. Además, los Web services se amoldan adecuadamente en cualquier escenario, en Internet (B2B & B2C), Intranets y Local Area Networks.

Hasta éste punto expuse las opciones de arquitecturas de sistemas distribuidos por separado, y en la sección 2.5 hice una comparación de sus características técnicas similares, y evalué la mejor opción.

En la sección 2.5 expusimos que la interoperabilidad de sistemas es una necesidad desde que se quiso compartir datos entre ambientes homogéneos. Así, se generó la incompatibilidad de tipos de datos y otras características entre ellos. Se atacó primitivamente este problema usando convertidores de protocolo, pero el problema con ellos era que representaban un monótono trabajo adicional para el equipo de desarrollo. Por tanto, la solución que se pensó fue hacer un protocolo universal, capaz de comunicar a componentes de una LAN interna entre sí y al exterior, hacia otra LAN. De hecho, éste protocolo universal conocido como Web services, hace innecesaria la tarea de generar convertidores de protocolo porque están construidos mínimamente sobre HTTP y XML. Los estándares universales permiten que los servicios de software sean publicados, localizados y utilizados en una forma que es independiente del lenguaje, plataforma y ubicación geográfica.

La evolución de los Web services revolucionó los sistemas distribuidos. Tienen éxito y aceptación debido a que la tecnología de Internet está basada en un sencillo conjunto de estándares. HTML y HTTP hacen posible el que las personas y los sistemas compartan información a través de Internet de una forma segura y confiable.

¿Por qué no han tenido tanta aceptación los protocolos como CORBA y RMI? ¿Por qué no existe un navegador CORBA comercial, o una Internet desarrollada no en HTTP sino en CORBA? Dichas arquitecturas no son ampliamente aceptadas porque la mayoría de las veces, se proponen soluciones para esas arquitecturas basadas en la conectividad para el puerto 80, pero ése puerto ya está reservado para aplicaciones Web. De la misma forma, tampoco es bien visto que los protocolos propietarios generen un túnel en el puerto reservado de HTTP, porque esta solución tiende a saturar las conexiones en el puerto, y además la programación se incrementa en complejidad.

DCOM, CORBA y Java RMI están diseñados para trabajar en un Datacenter Corporativo, no en Internet. Éstos protocolos son por herencia orientados a conexión.

En la sección 2.5 también mostré un cuadro comparativo con las tres arquitecturas que previamente desarrollé por separado, y al equiparar las características de una arquitectura con otra, encontré que los Web services ofrecen muchas comodidades al desarrollador de aplicaciones distribuidas, y entre muchas ventajas, determiné que se tiene diversidad en tipos de mensajes, tipos de transferencia, se tiene un esquema RPC simplificado, los mensajes son autodescritos gracias al lenguaje XML y extensiones como SOAP, WSDL y los mensajes pueden ser validados usando los XML Schemas.

El único inconveniente que saqué a la luz gracias a éste cuadro comparativo es que los protocolos de transporte para CORBA o RMI son más veloces que los protocolos basados en texto debido a que usan una codificación binaria de argumentos y valores de retorno. Esto se compensa con creces si se piensa en que SOAP utiliza XML para codificar los mensajes lo que los hace de fácil procesamiento en cada paso del proceso. La interoperabilidad a gran escala es el principal objetivo de la codificación XML SOAP. Además, los protocolos binarios representan un peligro en el caso de que un mensaje venga mal formado o contenga datos inválidos o en un orden incorrecto, y dichos métodos de codificación son pesados desde que requieren una compleja estructura para su correcto funcionamiento (no requieren uno sino varios servicios para procesar los objetos).

Los Web services pueden escoger de entre una amplia variedad de parsers de XML, debido a que dichos están disponibles para simplificar la creación y el consumo de documentos XML en prácticamente toda plataforma moderna. XML es ligero y es una excelente herramienta de soporte, por tanto, la codificación en XML permite un largo alcance debido a que prácticamente todo cliente en cualquier plataforma puede comunicarse con el Web service.

La sección 2.5 concluyó haciendo una reflexión de los costos y los beneficios de programar al puro estilo de XML sobre HTTP. En esta reflexión encontramos solo una mínima desventaja, en comparación a todos los beneficios que ofrece la arquitectura de Web services. También percibimos que las otras arquitecturas no tienen porqué despreciarse para otro tipo de desarrollos o escenarios. CORBA y RMI son indispensables en la resolución de otro tipo de problemas, mas no en el caso de estudio de ésta tesis. De hecho, ambos conceptos – CORBA y RMI- quedaron a nivel de "práctica de laboratorio", debido a que solamente probamos ambas arquitecturas en programas simples, y los Anexos 10 y 11 dieron prueba de ellos. De hecho, desarrollar una aplicación completa en tales arquitecturas es un digno tema para algún trabajo de investigación posterior.

En la sección 2.6 abordé el concepto de expresiones regulares de una manera formal, y expliqué que dicha nomenclatura es un patrón que debe seguir un conjunto finito o infinito de datos. Cité ejemplos de expresiones regulares. Además hice referencia a que las expresiones regulares tienen aplicación en el área de programación para construir analizadores léxicos o scanners.

La misma sección delimita el concepto de autómata finito, y los clasifica en determinísticos y no determinísticos. Los AFD tienen aplicación práctica en la validación de expresiones regulares. Aquí, mencioné la metodología para obtener el AFD a partir de una expresión regular, y expuse de qué forma puede programarse para validar una expresión regular de correo electrónico. Esta tarea de programación consta de tres rutinas, a saber, construir un arreglo, clasificar un carácter en una clase de símbolos y la última para validar la cadena dando seguimiento al arreglo.

En éste sentido, al escribir un ejemplo para validar una expresión regular de correo electrónico, mi intención es dar a conocer todo el trabajo que está detrás de validar cada uno de los tipos de datos, y señalar que los XML Schemas proveen la ventaja de que construyen automáticamente los Autómatas Finitos Determinísticos que usan, tan solo proveyéndoles la expresión regular como parámetro. De hecho, una aportación de esta tesis fue mostrar cómo se puede validar un XML SOAP Request con un XML Schema, puesto que al momento de escribir este trabajo de investigación, todavía hay poca información del tema, aún en Internet dicho tópico no es de publicación abierta.

Por otra parte, al habernos convencido de que la arquitectura de los Web services es la mejor opción en torno a los sistemas distribuidos, procedimos a analizar cada uno de sus componentes por separado en capítulo 3. Aquí, abordamos los conceptos de HTTP, XML y metodologías relacionadas, desde una óptica completamente orientada a servicio.

La sección 3.1 aborda el tema del protocolo HTTP. De hecho, de éste se ha escrito mucho, no solo en trabajos de tesis, sino en libros, manuales y cursos. Por tanto, en ésta sección solamente se justifica el uso de HTTP como medio de transporte, por ser ligero, veloz, apropiado para sistemas de información distribuidos y colaborativos. De hecho, el que HTTP sea genérico, sin estado y orientado a objetos, lo hace ideal para las aplicaciones residentes en Internet, debido a que dichas aplicaciones tienen la imperante necesidad de transferirse documentos entre ellas.

En ésta sección dejé establecido que HTTP ya no se usa puramente para HTML. Antes, a lo único que podía aspirar el usuario de Internet es abrir un socket, conectarlo con PPP y SLIP, montar una pila de TCP/IP sobre esos protocolos y transferir puras páginas de HTML a su computadora, como libros, manuales, noticias y cosas sencillas. Ahora HTTP es utilizado para establecer un puente entre componentes de software y crear servicios, sin despreciar el camino recorrido de la tecnología en torno a éste protocolo.

HTTP tiene muchas ventajas, de hecho resuelve el problema de que es un protocolo de transporte que puede usarse igualmente para ambientes de LAN como para Internet, siendo esto imperceptible para el usuario. HTTP rompe barreras técnicas y humanas, al ser aceptado por los administradores de redes modernas.

En ésta sección detallé la estructura de URL, ejemplos de solicitudes y respuestas HTTP, y el hecho de que el método POST es ideal para la transferencia de información como documentos XML.

La sección 3.1 concluye citando el uso de un encabezado especial llamado SOAPAction, y mostrando cómo se vería un XML para solicitar un servicio, y la respuesta que daría dicho servicio, también en formato XML. Tal ejemplo ilustra el concepto de Web services de una forma sencilla pese a que introduje prematuramente la gramática SOAP.

La sección 3.2 introduce los elementos esenciales del lenguaje XML, cuyo propósito es estructurar datos. Los datos estructurados son datos que tienen relaciones internas bien definidas y restricciones que proveen un contexto para los datos, la mayoría de las veces imprescindible para su correcto procesamiento. En ésta sección mostramos la importancia de migrar nuestros pensamientos a este nuevo paradigma, y abandonar de una vez por todas los conceptos tales como diccionarios de datos y layouts para interpretar datos de una plaza en otra, y ajustarse a los estándares en la búsqueda de satisfacer la necesidad de compartir datos estructuralmente entre organizaciones. XML ha tenido un vasto éxito en el comercio electrónico negocio a negocio.

Además, mostramos que el instrumento utilizado por XML son los metadatos, es decir, datos que describen otros datos. El concepto metadatos es indispensable si se parte del hecho de que los datos son útiles si su contexto es comprensible.

Sin XML, las organizaciones enfrentaban problemas que ocasionaban pérdida de productividad e inflexibilidad del negocio. XML permite muchas operaciones sobre la información, entre ellas distribuir información, generarla en múltiples formatos, integrarla desde múltiples fuentes y administrarla y combinarla condicionalmente con un bajo tiempo de respuesta.

XML se apega a las necesidades de información de las empresas de la actualidad, porque es una sintaxis de marcas o etiquetas que define la estructura de los datos de una forma abierta y autodescriptiva. XML hace que los datos puedan ser tanto transferidos exitosamente de las aplicaciones generadoras a las receptoras como procesados consistentemente. XML y HTML son semejantes en el sentido de que son protocolos, formatos y archivos basados en texto, lo que los hace fáciles de procesar por cualquier aplicación. La diferencia es que XML es utilizado para enviar datos de un lugar a otro sin formato, mientras que HTML se encarga de presentar esos datos para un ser humano, dandoles formatos visuales, pero esos formatos son inútiles para que una aplicación receptora interprete correctamente la información.

En la misma sección 3.2 mencioné que los documentos XML contienen tres tipos de información al mismo tiempo: los datos, su estructura y su presentación. De esta manera el lenguaje ha permitido un mejor método de comunicación entre aplicaciones. También permite que diferentes tipos de dispositivos se conecten a la Web.

Generar XML permite que los web servers determinen que tipo de dispositivo realiza la solicitud y responder utilizando el formato de datos adecuado.

Los navegadores han avanzado en la tecnología de XML, porque ahora éste tipo de programas visuales manejan los datos de formas “inteligentes”, es decir, traducen las etiquetas significativas, y pueden ser alteradas y consultadas con scripts sencillos del lado del cliente.

En ésta sección destaqué que, si bien HTML provee un conjunto de etiquetas para dar formato a los datos, XML se encarga de describir los datos de forma significativa, para que una aplicación se los envíe a otra a través de la web.

XML hace fácil intercambiar información dentro de una misma organización o entre diferentes organizaciones. Esto se le conoce como integración de diversas fuentes de información para presentar un producto final de datos en una interfaz uniforme.

XML tiene ventajas y desventajas, pero las desventajas son relativas si se piensa que ya existen soluciones elaboradas para tratar conceptos como son namespaces y normalización de estándares.

Se eligió XML como parte de la solución al caso de estudio debido a que es tan versátil que su uso es cada vez más extenso, no solo sirve para integrar las aplicaciones empresariales, sino que también es utilizado en el comercio electrónico negocio a negocio, y para distribuir información sin tener que trabajar mucho para formatearla como se desee.

En la sección 3.2 consideré los conceptos técnicos del lenguaje XML, como son, los elementos, las declaraciones, las instrucciones de procesamiento, los comentarios, los elementos hijos, los elementos vacíos y los atributos. Todos estos elementos están organizados en dos partes del documento, que es el prólogo y el elemento root. Además, todo lenguaje tiene una gramática, o un conjunto de reglas a seguir, y XML no es la excepción, de hecho, mencioné reglas específicas para saber si un documento está bien formado.

La mayoría de las herramientas basadas en XML realizan el "marshall" o composición del documento XML de manera que éste quede bien formado. No obstante, algunas veces ese documento es formado "a mano" o con herramientas de programación no orientadas a objetos, por lo que llegan a tener errores en la estructura, datos o jerarquía. Para tal situación definí que los XML Schemas y los DTDs pueden usarse para garantizar que el documento cumpla con los lineamientos de la gramática.

Aquí definí que una gramática XML es un conjunto de elementos y atributos preestablecidos que son típicamente representados utilizando un DTD o un Schema. De hecho, existen tantas gramáticas como tareas o mercados. Las gramáticas utilizadas en éste trabajo de investigación son la de los XML Schemas (XSD), Web Services Description Language (WSDL) y Simple Object Access Protocol (SOAP). De esta manera una gramática restringe perfectamente el contenido de una familia de documentos XML. En este caso, se dice que dichos documentos son instancias de la gramática definida en el XSD o XML Schema.

En la misma sección mostramos cómo los namespaces son los nombres que reciben las gramáticas. De hecho, antes de usar un namespace es necesario declararlo. Pueden declararse múltiples namespaces en un mismo documento. También puede darse el caso de que se tengan namespaces por default en elementos y atributos y habrá casos en que habrá que declararlos explícitamente. Al analizar el papel de los namespaces en la definición de las gramáticas, es posible comprender el trasfondo y la filosofía del lenguaje XML, la cual establece que en un documento se pueden tener elementos provenientes de diversas gramáticas, y pueden incluso ser validados a partir de las reglas de sus gramáticas de origen. Los namespaces son conceptos esenciales para comprender XML y sus tecnologías relacionadas

Al final de la sección 3.2 dimos un par de referencias para estudiar namespaces, gramáticas y XML Schemas. De transfondo, puede decirse que las tres cosas significan exactamente lo mismo, por lo que invitamos al lector a seguir estudiando este concepto por su cuenta hasta que le quede 100% claro, ya que sin esta herramienta es imposible desarrollar XML Web Services.

En la sección 3.2.1 toqué más a detalle cómo los XML Schemas pueden describir otros documentos XML. En vista de que es posible crear nuevos tipos de elementos y atributos para satisfacer las necesidades de determinado tipo de documento, una asociación de entidades podría acordar una gramática XML que defina un conjunto compartido de elementos y atributos que pueden ser utilizados para un tipo de documento en particular. En este sentido, la gramática XML especificó criterios en torno a un conjunto de elementos, su orden, sus relaciones permitidas entre ellos, el número máximo y mínimo de veces que un elemento puede presentarse, y la lista de atributos permitidos para cada elemento. Por consiguiente, los documentos XML experimentan validación para revisar que cumplen las reglas de una gramática particular. La validación es entonces un magnífico criterio de discriminación de mensajes inválidos y permite que una aplicación u

organización receptora determine si un documento es una instancia legítima de su gramática, definida previamente.

Los XML Schemas son una especie de contrato en el que dos o mas partes acuerdan la estructura de los documentos XML de instancia que serán procesados automáticamente por un nuevo servicio. De hecho, cada instancia de solicitud de transacción debe adherirse perfectamente a dicho formato.

La aplicación mostrada en el capítulo 4 se amolda correctamente a este paradigma, porque cada vez que uno de los distribuidores envía un reporte de ventas y orden de reparación, el corporativo procede a revisar la estructura de la información contra el XML Schema definido en los estándares. Además, el corporativo revisa que los datos contenidos en el documento no violen las restricciones de tipos de datos impuestas por el Schema. De hecho, si llegaran 40 mensajes al mismo tiempo al Web service, los mismos 40 serían revisados simultáneamente.

La sección 3.2.1 mostró todos los lineamientos a seguir para crear la estructura de un XML Schema. Además, concluí en esta sección que existen dos formas de validar un documento de instancia contra un documento XML Schema, la via “manual” o usando un plugin de un navegador, y la via automática, o a través de una biblioteca especializada en procesamiento de XML. Aquí consideré las razones para asegurar que es mucho mejor la validación automática que la vía manual.

La recomendación de W3C para los XML Schemas incluye un número de tipos de datos listos para usarse que corresponden a tipos comunes encontrados en lenguajes de programación y sistemas manejadores de base de datos.

La sección 3.2.1 nos sirvió para mostrar cómo declarar un elemento o un atributo en un XML Schema dando la sintaxis de creación exacta que permite hacer esta tarea. Por otra parte, debemos decir que los XML Schemas es nuestra parte favorita del lenguaje XML, debido a que a través de ésta gramática pudimos definir reglas o restricciones en la gramática acerca del tipo de datos. Los nuevos tipos de datos pueden derivarse a partir de tipos de datos existentes. Esta derivación puede venir definida a través de una o más reglas o restricciones. A nuestro parecer, la restricción “pattern” es la más útil porque especificamos una expresión regular con la cual debe cumplir un tipo de datos dentro de un documento XML. También se mostró cómo crear diferentes tipos de datos complejos, que son la combinación de otros elementos, atributos o datos especiales.

Los XML Schemas tienen múltiples características de su naturaleza por herencia orientada a objetos, tales como hacer referencias a declaraciones existentes y de ésta forma reutilizar estructuras dentro de los XML Schemas. Esto permite que no se esté reescribiendo una estructura cada vez que se utilice.

En la sección 3.2.2 dijimos cómo XML es utilizado para el intercambio de información. En ésta sección hablamos sobre un caso particular de los documentos XML, la gramática XML SOAP, la cual es utilizada para pasar los parámetros por valor o por referencia a un servicio, y de la misma forma recibir una contestación exitosa o fallida. Aquí, vimos que los documentos están compuestos por tres elementos básicos: Envelope, Header y Body. Las respuestas de tipo fault están compuestas por un faultcode, falutstring, faultactor y detail, aunque en el servicio java para la industria automotriz no fue necesario utilizar la sección detail. XML SOAP es versátil en el sentido de que puede transmitirse este tipo de documentos a través de HTTP, SMTP o cualquier otro protocolo diseñado para el transporte de información.

Por otra parte, SOAP es típicamente usado para transferir un mensaje de un punto a otro, pero esto no es siempre el caso. Puede darse el caso que un mensaje pase por las “manos” de varios intermediarios antes de llegar a su destino final. Dichos intermediarios ejecutan alguna operación sobre el mensaje antes de retransmitirlo al siguiente actor o al destino final. Dichos intermediarios son conocidos como “SOAP Actors”.

Estrictamente hablando, no es obligatorio validar un mensaje SOAP antes de procesarlo, pero, la empresa corporativa regularmente lo hace y lo seguirá haciendo, cuando se recibe información delicada como son transacciones monetarias por compra/venta, financiamiento, transferencia de fondos o bienes, y operaciones relacionadas.

Al llegar a la sección 3.2.3 señalamos cómo el XML es utilizado para describir servicios también en formato XML. Establecimos que WSDL es otra gramática derivada de XML y de los XML Schemas, a fin de describir abiertamente un servicio de software en Web. En ésta sección declaramos que las etiquetas básicas de WSDL son definitions, types, message, portType, binding y service, cada uno con una tarea específica y muy directa para componer el contrato electrónico de uso de web services. Además, definimos un diagrama jerárquico de dependencias funcionales, en donde se muestra como un elemento está encadenado con otro. A partir de esta información es como se construyeron los WSDL para ventas y servicio del Anexo 8.

En la sección 3.3 hablé sobre los métodos de localización de los Web Services, el principal de ellos UDDI, quien provee un servicio de directorio central para publicar información técnica acerca de los Web services. UDDI todavía no es un estándar adoptado por la W3C, aunque si es producto de la iniciativa privada de empresas tecnológicas de la talla de Microsoft, IBM y Ariba.

En dicha sección califiqué a UDDI como la infraestructura que está compuesta por “registries” y “registrars”. Los registries contienen copias completas de un directorio UDDI, mientras que un registrar provee un servicio de registro de UDDI, en representación del usuario de Web services. El objetivo de UDDI es almacenar la información de diferentes Web services, para que los clientes puedan de manera dinámica determinar que Web service utilizarán en base a disponibilidad, precio y características, todo ello sin la intervención de la mano humana.

En el caso de la industria automotriz, aún no utilizamos ningún tipo de registry, por lo que este tema puede quedar propuesto para próximos trabajos de investigación, o la continuación de éste mismo trabajo, registrando el Web service en dichos UDDI, para después buscarlo dinámicamente con las herramientas diseñadas para ello.

En la sección 3.3 también expusimos los diferentes métodos de consulta y publicación que puede tener un UDDI registrar. Es de esperarse que un Web service utilice todos los métodos de publicación, y un Web Client utilice todos los métodos de consulta.

Considero que la sección 3.4 fue un “bind all together now”, es decir, mostré cómo todos los elementos de Web services trabajan armoniosamente, en conjunción, para garantizar el envío síncrono y asíncrono de XML sobre el tradicional protocolo HTTP para transferencia en cualquier tipo de red. Lo sobresaliente de ésta sección fue saber cómo los Web services funcionan en el comercio electrónico. También clasifiqué a los web services de dos maneras: por su tipo de solución (Application Service Providers & Hosted Applications vs Application Integration) y por su tipo de arquitectura(orientados a RPC o síncronos y orientados a mensaje o asíncronos). Para todas las clasificaciones, es de notar que una no es mejor que otra, sino que sencillamente cada una de ellas tiene un escenario ó tipo de aplicación en el que se desenvuelve. Hay que conocer las dos caras de cada moneda que llegue a nuestras manos.

Concluí la sección 3.4 con el análisis de las arquitecturas orientadas a servicio, la cual consta de un proveedor de servicios, un consumidor de servicios y un publicador ó kiosco de servicios, conocido como “broker” o integrador. En este sentido, se determinó que la arquitectura de Web services es orientada a servicio debido a que se tienen los elementos correspondientes en tal arquitectura, llamados XML Web service provider, XML Web service consumer, y XML Web service broker, respectivamente. Además esquematizé a detalle las interacciones que tienen estos tres elementos entre sí, y la eventual necesidad de un broker.

El capítulo 4 está enteramente dedicado al caso de estudio, porque se muestra como las herramientas conjugadas –la base de datos mysql, el lenguaje java y los web services - forman una solución a la situación problemática cimentada en el capítulo uno de la tesis.

El capítulo 4 abre con la sección 4.1 la cual establece exactamente una hipótesis. Resumidamente, se puede decir que el diseño de la base de datos se tocó a detalle en la sección 4.2. La creación de la interfaz para efectuar transacciones en el sistema, los métodos para ventas y para servicio se explicaron en la sección 4.3 y

el diseño de las reglas de validación de tipos de datos se mostró en la sección 4.4. Finalmente, el diseño y la utilización de la interfaz de Web services usando MSSoap Toolkit se mostró en la sección 4.5

En la sección 4.1 propusimos un proceso, una respuesta, una investigación aplicada, la que hace una pequeña aportación a la comunidad científica sobre una aplicación concreta de los Web services. De hecho, en ésta sección reproducimos el diagrama del capítulo 1 y mostramos la solución implantada que da respuesta a las necesidades de información del mercado de trabajo del corporativo automotriz. Para no dejar la interpretación del diagrama al criterio de cada ingeniero, establecimos una narrativa de las expectativas reales que esperamos del sistema, o dicho mas propiamente, de la interfaz. Dicha interfaz la construimos con tres elementos nuevos: un cliente de MSSOAP Toolkit, el Java Web Services (que contiene tanto el Tomcat Application Server como las Java classes) y los servicios de base de datos MySQL.

Dentro de la base de datos explicada en la sección 4.2, concluimos que los reportes de ventas y de servicio están íntimamente relacionados, al grado que deben compartir el mismo espacio de trabajo en el servicio del motor MySQL. De hecho, se tiene que se comparten 6 tablas entre los dos tipos de reportes. Como comentario, la política de la empresa automotriz establece que 'el cliente es primero' por lo que es vital tener información de clientes tanto de ventas como de servicio, porque se espera que un cliente de ventas se convierta en cliente de servicio. Si se conoce al cliente, existe mayor probabilidad de atraer ingresos a la compañía a través de sus distribuidores.

Para crear una secuencia de comandos SQL de tipo Data Definition Language, definí "niveles de creación" para las tablas, es decir, dependencias funcionales dadas por las relaciones 1 a 1 y 1 a n dentro del sistema, y fijé en el nivel 1 todas las tablas independientes, es decir, que no dependen funcionalmente de nadie. Acto seguido, mostré las llaves primarias en el sistema, así como su justificación de cada una de ellas.

Concluí la sección 4.2 mostrando las bibliotecas almacenadas en el package coreservlets, las cuales administran las conexiones de la base de datos de manera concurrente, por lo que se pueden administrar 50 conexiones entre los 150 distribuidores en la república mexicana, lo que se tiene un ahorro considerable tanto en memoria como en pesos, a futuro.

Una vez que completamos el diseño y la normalización de la base de datos, escribimos el SQL con instrucciones DDL e instalamos estructuras relaciones y catálogos en el servidor de MySQL, copiamos los manejadores de JDBC al servidor y procedimos a mostrar la metodología para diseñar, programar y utilizar la interfaz de web services, dando a conocer los factores involucrados en la programación del servicio.

En la sección 4.3 introduje el hecho de que la base de datos puede ser consultada directamente por el corporativo, sin intermediarios, usando quizás un sencillo conector como Microsoft ODBC para MySQL –qué a su vez provee la fuente de datos para aplicaciones del Microsoft Office 2003- , o bien el cliente gráfico de MySQL. No obstante, no asignamos al distribuidor una cuenta en la base de datos para que él hiciera las transacciones, y de esa forma "ahorrarnos" trabajo, debido a que el servicio de base de datos está situado en el puerto 3306, cerrado tanto para la red corporativa como para la red local del distribuidor. En este sentido, se tendría que instruir al distribuidor en materia de SQL, cosa que es absurda ya que la principal tarea del distribuidor es estar al tanto de su negocio y utilizar los sistemas, no programar en un lenguaje de SQL. Además, expuse el hecho de que el distribuidor no sabría como reaccionar ante un cambio de motor de base de datos, ni habría forma de administrar un número limitado de conexiones entre el número real, y variable, de distribuidores. En éste sentido, la base de datos estaría sujeta a intensos ataques y si la base generara un error (que generalmente está etiquetado con números) el distribuidor no sabría que medidas correctivas y preventivas tomar.

Estos problemas y demás los abatimos con un agente especial capacitado para comunicarse por el puerto 80 u 8080, que hable SQL por el distribuidor, que maneje la base de datos de forma que para el usuario sea transparente, que administre un número limitado de licencias, protegido con varias capas de seguridad, que provea mensajes descriptivos de error, y además que se comporte adecuadamente en redes de baja velocidad de transferencia y que use protocolos universales como XML SOAP, y que ejecute tareas de validación sobre las transacciones a la base de datos. Ese agente lo definimos como un todo, en el bloque "Java Web Services".

Después de sustentar la necesidad de tal servicio, procedí a describir la arquitectura RPC de dicho bloque que da respuesta a cada solicitud XML SOAP. Dicho simplemente, el distribuidor envía su solicitud en un sobre y espera la respuesta, al recibir la respuesta, el distribuidor abre el sobre y revisa si en él hay una respuesta exitosa o errónea.

Después de mostrar la arquitectura, generé un conjunto de transacciones de ventas como para servicio, en suma 19. Las operaciones de ventas son AltaUnidad, BajaUnidad, AltaInventario, AltaVendedor, BajaVendedor, ListaVendedores, AltaCliente, BajaCliente, ListaClientes, ReportaVenta, CancelaVenta y Transferencia. Las operaciones de servicio son AltaCliente, BajaCliente, ListaClientes, AltaAsesor, BajaAsesor, ListaAsesores, AltaOrden, ImportesOrden, CierraOrden, CancelaOrden. En todas las transacciones incluí información del usuario y del password en cada transacción. Además, antes de que el sistema realice una operación sobre la base de datos, recurre a tres niveles de reglas de validación, que son, estructura y tipos de datos de xml, reglas de negocios y las reglas de integridad referencial dadas por la base de datos.

Debido a que todas las transacciones requieren un específico conjunto de parámetros, los documentos XML recibidos en la mensajería HTTP son transformados en árboles DOM, que pueden tener un número fijo o variable de nodos. En ésta sección hicimos la clasificación pertinente.

Dentro de los métodos de Web Services, los métodos ListaAsesores, ListaClientes y ListaVendedores son de carácter informativo en la agencia distribuidora, Es menester tener un claro conocimiento de éstas entidades asociadas a la distribuidora, ya que sin ellas no es posible reportar ventas ni órdenes de reparación.

En ésta misma sección introduje representaciones gráficas de los objetos Java de tipo DOM Tree y definí el concepto de descomposición ó unmarshalling o transformar un documento XML en un DOM Tree. Por el contrario, el marshalling convierte un DOM Tree hacia un documento XML.

El unmarshalling es indispensable en la programación ya que un DOM Tree es mas legible que el XML puro. De hecho, mostré en los programas del Anexo 5 que sin el unmarshalling no puedo recuperar los parámetros de la solicitud, y sin el marshalling, no puedo enviar una respuesta de regreso al distribuidor.

Dentro de los documentos XML SOAP Request y XML SOAP Response observé que los nodos Envelope y Body pertenecen al namespace oficial definido por la W3C para los XML Schemas, mientras que los nodos definidos a título personal no cuentan con namespace alguno, para facilitar la programación de la interfaz PROXY, aunque no habría ningún inconveniente en definirles su propio namespace si los estándares de la industria automotriz cambiaran el día de mañana.

En ésta sección noté lo útil que es visualizar los documentos XML en un navegador, ya que éste provee controles para extender y comprimir los nodos de XML, y hacer legible y manejable cualquier documento, por grande que sea. Al observar mis documentos XML de ejemplo con el navegador, el lector notará que tanto las solicitudes como las respuestas están codificadas en el lenguaje XML, con la gramática particular de SOAP.

Después que asigné los documentos XML al intercambio de información entre los Web services y los clientes, y los servicios de base de datos quedaron adscritos exclusivamente para el Java Web Server, descompuse la solución para revisar su estructura órgánica en un diagrama que titulé “Java Virtual Machine”, porque el cliente en realidad se está comunicando con una JVM de Java sin notarlo siquiera, ya que ésta provee un servicio HTTP en el protocolo TCP/IP. De forma narrativa expuse el ciclo de vida de una transacción, y mencioné el tipo de bibliotecas de software que se usan en esa trayectoria. Además mostré cómo están construidas las clases de aplicación. De hecho, estas clases pueden agruparse y definir una superclase entre ellas, debido a que su apariencia y funcionalidad es muy semejante, variando solamente en puntos muy concretos.

En la sección 4.4 hice notar que es indispensable que la información sea valida antes de llegar a la base de datos, por tratarse de datos críticos para la compañía. Para éste caso, coloqué un “agente consular” en la entrada del servicio web para que revise cada una de las solicitudes de todos los métodos. En este sentido dibujé un diagrama que muestra a las clases recibiendo no solamente las solicitudes de XML SOAP, sino

también los XML Schemas del servidor vía HTTP. De éste diagrama concluí que los XML Schemas vienen almacenados en archivos XSD, y tienen directivas almacenadas para el agente de validación. Los XSD son pequeños pero eficientes manuales de políticas y reglas que le recuerdan al usuario de Web services las políticas y reglas de la industria automotriz. En ésta sección di a conocer además que el problema no es escribir el agente de validación quien se deriva de clases muy específicas, sino mas bien, escribir correctamente el XSD. En ésta sección cité ejemplos de los XML Schemas que utilice y qué pasos dí para usarlos en la validación de las familias de documentos XML SOAP Request. Los XML Schemas son lenguajes de descripción de lenguajes o metalenguajes, y son los medios formales de descripción de documentos XML y son el punto de partida para validar todos los SOAP Requests que reciben los Java Web Services.

En ésta sección aclaré que tengo un XML Schema para cada método transaccional en el servicio.

Concluí la disertación de la sección 4.4 señalando los beneficios de haber utilizado los XML SChemas en la validación de campos fecha y datos definidos por nosotros, como el rfc, el número de serie del vehiculo y las claves de acceso al servicio. De hecho, alterar expresiones regulares no implica recompilar nada. En éste sentido, los XML Schemas validan tipo, orden, relación y estructura en los datos. Si el mensaje aprueba la revisión, pasa al la siguiente etapa de validación, en caso contrario, el mensaje se desecha y se notifica al distribuidor la causa.

La sección 4.5 la dedicamos a contestar la pregunta de cómo utilizar el servicio, o cómo construir un cliente ligero para adaptarse como componente en el lado del distribuidor, indicando que no es una carga pesada crearlo. También encuadré los elementos para elaborar el cliente, tales como una base de datos relacional pequeña, un lenguaje de programación amigable con la base de datos, y bibliotecas para manejar XML y transmitirlo por HTTP. Todos pueden reunir mis requerimientos para elaborar su propio cliente, no importando si poseen PCs de escritorios, servidores, terminales o estaciones de trabajo. Aquí requiero que el cliente tenga una pequeña base de datos no obligadamente, pero sí sugeridamente, para que también lleve un pequeño registro de las transacciones que va efectuando en el sistema.

En la industria automotriz en que laboré, los proveedores desarrollaron clientes para el Web Service de todo tipo de formas y apariencias, pero, antes de que cada proveedor certificado de software en las agencias distribuidoras eligiera el motor, el lenguaje y la librería de su preferencia, el departamento de enlace al distribuidor debía poner el ejemplo al desarrollar una aplicación que mostrara el uso de los Web services. Para obligar a que el distribuidor siguiera el modelo, se le dio a este programa una fecha de expiración, y se dieron presenciales para enseñarle al distribuidor cómo elaborar su propio cliente de Web services.

En ésta sección aplicamos la metodología del capítulo 4 para escribir pequeñas rutinas capaces de establecer el Proxy del contenedor SOAP, y de ésta manera convertir una llamada a procedimiento local en un mensaje viajero de XML.

Continué mi análisis con mi versión personal del cliente de Web services, el cual fui probando con cada uno de los métodos, primero de ventas, y luego de servicio, auxiliándome en “screenshots” o pantallas de captura, para indicar paso a paso cómo utilizar el servicio de una forma adecuada. Apoyé dichas acciones consultando directamente la base de datos para verificar que efectivamente el mensaje XML se transformó en una sentencia SQL ejecutada en el servicio de MySQL.

Dentro de la sección 4.5 hicimos un pequeño apartado 4.5.1 para mostrar los errores más comunes que pueden presentarse en la aplicación y cómo éstos deben ser resueltos. De hecho, si un error no aparece en la lista de los “20 más buscados” que definí en ésta sección, puede utilizarse el servicio espejo InstanciaSOAP.java para obtener una copia exacta del XML SOAP Request que está tratando de enviarse al servidor, y en el peor de los casos, consultar qué línea de código está fallando en el Anexo 5.

3. Prueba de hipótesis

La meta de la industria automotriz es la misma que todas las organizaciones cuya actividad es comercial y lucrativa: producir dinero. La industria del transporte terrestre lo hace a través de la venta de unidades, refacciones, servicios y franquicias. Todo mundo sabe que entre mas clientes finales tenga la cartera de una compañía, existen mejores probabilidades de vender estos productos.

Después de implantar los Web services en el corporativo y las agencias, se eliminó el riesgo de tener un dato inexacto de cuantas unidades nuevas se vendieron a los clientes en el mes, o cuantas órdenes de reparación se levantaron en el mismo periodo de tiempo. Si el corporativo desconoce cuales y cuantas unidades vendió a través de sus filiales, no hay forma de reponer tales unidades físicamente, ni tampoco hay manera de determinar hacia donde se están orientando las ventas, o qué modelos se venden en qué temporadas. De la misma forma, si el corporativo desconoce qué tipos de reparaciones se efectúan en las distribuidoras, le será muy difícil detectar defectos de fabricación en las unidades nuevas, y además determinar qué tan satisfecho se encuentra el cliente final con su unidad seminueva o con el servicio que se le brinda.

De hecho, éste es el problema que atacó la hipótesis. Se trabajó en tener datos más exactos sobre el número de órdenes de reparación y el número de reportes de ventas de todos los distribuidores, para detectar las distribuidoras que envían escasamente ésta información, y para inspeccionar las que envían regularmente a través de una interfaz de Web services.

El que los distribuidores adoptaran los servicios que el corporativo ofrece a través de un servidor Web hizo que tuvieran beneficios adicionales, como es que el corporativo publicara su inventario en línea, en Internet, para posteriores ventas en línea con apartado de tarjeta de crédito. Además, el mismo sistema se encarga de registrar en su base de datos los movimientos de transferencia, para auditar quién está adjudicandose unidades que por derecho le corresponden, y quién está haciendo transferencias ilegales de unidades al inventario de su distribuidora, tan solo para alcanzar objetivos de ventas, para después enviar la cancelación en el próximo mes, asunto que era delicado y no se llevaba control del mismo.

Por otra parte, auditar la calidad de los datos que envía el distribuidor garantiza la utilidad que se le da a la información porque de otra manera, cuando un distribuidor envía un dato no válido, el corporativo no se molesta en tratar de descifrar qué es lo que quería transferir el distribuidor, mas bien, desecha los datos inválidos hasta que el distribuidor los corrija. En nuestros días, están dejando de existir los sistemas de información que no validan los datos antes de recibirlos. La tendencia es que nadie se arriesga a recibir información espuria.

Además, éste trabajo de investigación hizo mucho hincapié a la importancia de validar cualquier dato que se recibe de una fuente externa, por ejemplo de los distribuidores a su corporativo automotriz. Comenté que validar datos puede hacerse de diversas formas. Por ejemplo, los lenguajes de programación incluyen funciones destinadas para, revisar la longitud de una variable, conocer si la variable es numérica, de tipo carácter o de tipo fecha. Otros lenguajes de programación contienen métodos para saber si un objeto de tipo String contiene una secuencia de caracteres inválidos. Estas herramientas ligeras son adecuadas para comprobar tipos de datos simples, pero cuando se incrementa el número de condiciones y restricciones que debe obedecer una expresión, las funciones y métodos preconstruidos resultan insuficientes, por lo que es necesario utilizar otras opciones.

Por tal razón, esta tesis fue un excelente vehiculo para divulgar que, según la experiencia académica y profesional, esas opciones para validar tipos de datos con un elevado número de condiciones y restricciones son los autómatas finitos determinísticos (AFD). Un AFD es capaz de validar cualquier tipo de datos en el tratamiento de información (intercambio, captura, transferencia, etc.). Los autómatas finitos determinísticos son los motores de los analizadores léxicos que a su vez se construyen a partir de una expresión regular. De hecho, un compilador tiene uno o más analizadores léxicos, sintácticos y semánticos, y tienen el objetivo de transformar un programa de un lenguaje a otro. De lo anterior se puede asegurar que los analizadores léxicos sirven tanto para validar un programa antes de convertirlo a otro, como para validar un dato antes de almacenarlo en una tabla. El analizador léxico es entonces el subconjunto de un compilador.

En éste trabajo de investigación dejamos establecido que un AFD no esta supeditado a la construcción de compiladores sino que hemos probado con agrado que el análisis léxico y la búsqueda de tokens o símbolos dentro de una secuencia de bytes o caracteres tienen aplicación en la validación de datos. Además, lejos de escribir rutinas para construir AFDs de expresiones regulares de nuestro interés, dejamos que los XML Schemas hicieran la tarea en nuestra representación al usar etiquetas como restriction y pattern. Los XML Schemas permiten tener un mecanismo de mejora contína al estar investigando de forma constante cual es la expresión regular que mejor describe las condiciones y restricciones de un dato, reglas que son necesarias para aplicar en un sistema.

Estas fueron básicamente las razones para proteger el Web service con la validación de los XML Schemas. Regularmente quien envía información inválida es el personal de reciente ingreso. Pero los XML Schemas solamente se ocupan de rechazar información no permitida. Esto se irá eliminando en la medida que al nuevo personal se le capacite de manera conceptual dentro del modelo de los Web services, ya que ésta tecnología como tal, todavía no se enseña como una asignatura a nivel técnico o profesional en todas las escuelas, sino que se imparte como taller o materia solamente en cursos muy especializados.

Los Web services son la mejor opción para aplicaciones distribuidas sobre Internet y sobre redes típicas basadas en TCP/IP y HTTP. En suma, el capítulo 5 arroja resultados para argumentar que se alcanzaron los objetivos de la tesis, ya que se resolvió la situación problemática y se verificó que la solución propuesta era una buena respuesta, al dejar atrás información extemporánea con procesos en línea, al eliminar la recaptura de información en las distribuidoras, al validar los datos recibidos en el corporativo, al cumplir con las normas corporativas de seguridad en red transfiriendo información en HTTP, al manejar datos autodescritos con XML, al ser una buena inversión de tiempo y recursos materiales, al trasladar un sistema heterogéneo de muchos proveedores certificados de software en uno homogéneo, al manejar adecuadamente la información de ventas, inventario, transferencias y servicio, al utilizar estándares conocidos y tecnología de punta y al demostrar que la solución aporta un nuevo volumen de datos reportado por las distribuidoras. Todo esto fue posible gracias al manejo de los Web services, resultados que dudamos haberlos alcanzado usando otras arquitecturas de objetos distribuidos como RMI y CORBA.

El capítulo 6 del presente trabajo de investigación cierra con una reflexión de alta importancia, es decir, que la industria automotriz no solo debe estar preocupada por tener completa la información de sus variables críticas de ventas de unidades nuevas y servicio a unidades seminuevas. También sería prudente que maneje gráficas de control estadístico para vigilar contínuamente dichas características de calidad. Las gráficas de control son elaboradas a partir de la selección de un grupo racional, de la reunión de los datos necesarios, el cálculo de la línea central de ensayo y los límites de control, la revisión de la línea central y los límites de control, para inalmente alcanzar el objetivo. Éstas gráficas de control, lejos de ser un trabajo innecesario, tendrían el único propósito de mejorar la calidad del desempeño de los distribuidores, definir la capacidad del proceso, tomar decisiones relativas a las especificaciones de los productos vendidos, tomar decisiones relacionadas con el proceso de la producción y tomar decisiones relativas a productos recién elaborados. De hecho, éste tema es propuesto para tratarse de mejor manera en otros escritos.

4. Reflexiones finales

El sistema presentado en el presente trabajo de investigación es adaptable para cualquier industria automotriz. Puede modificarse para que cumpla con las reglas de negocios de la industria que lo adopta e implementa. De hecho, los Web services tienen mucha utilidad, y si ayer tenían mucho potencial, hoy su utilización es cada vez más extensa, no solo en aplicaciones para la industria automotriz. En sistemas, los Web services hacen sentir la libertad parecida a la que se sintió cuando se inventó la telefonía inalámbrica en las telecomunicaciones porque éstos se adaptan a cualquier caso que implique transacciones de la forma solicitud-respuesta. Por los resultados obtenidos en ésta tesis puede concluirse que es la mejor opción para integrar sistemas en Internet o Intranet, con el objetivo de compartir datos entre sistemas muy diferentes entre sí, haciendo uso de estándares conocidos y nuevas tecnologías.

En éste trabajo de tesis, aunque no se uso CORBA o RMI para desarrollar la entera interfaz de sistemas distribuidos, se utilizaron los conceptos como stub, skeleton y proxy que establecen un contrato de transferencia

de datos entre entidades o componentes no necesariamente iguales entre sí. Pero esa no es toda la utilidad de estas arquitecturas. RMI y CORBA no necesariamente han sido totalmente remplazados por los Web services. Existirán casos en que se quieran publicar componentes de Java en la red corporativa, o bien reutilizar un viejo componente de Cobol, C, Fortran o Algol con los nuevos lenguajes de programación, sin tener que reprogramar todo el componente.

Dentro de los Web services todavía queda mucho por hacer. TOMCAT cuenta con muchos recursos que no se utilizaron en la tesis. Entre ellos están los conectores de bases de datos, los Java Beans, las Java Faces, los conectores tipo HTTPS, JAXB (librería de automatización de registros), etc. Quizás en posteriores proyectos empecemos a utilizar estas herramientas para seguir mejorando la calidad, la funcionalidad y la sencillez de la aplicación.

Otro rasgo que se puede considerar en futuras versiones del Java Web Service presentado es proveer hojas de estilo (en archivos XSLT) para compilar documentos XML y transformarlos en tablas de HTML de manera muy sencilla. Además, otro de los objetivos pendientes podría ser mostrar cómo construir clientes de Web services con Java, C++, Xerces, etc., con la intención de probar que no solo existe interoperabilidad Java/Microsoft, sino también Java/Java, Java/Xerces, Java/C++, etc.

Otra cosa que bien podría preceder a éste trabajo es proponer Web services que manejen información de refacciones, vehiculos seminuevos, étc., así como registrar los Web services de ventas, transferencias, inventario y servicio en un UDDI; perfeccionar los archivos provistos en el WSDL del Anexo 8, y utilizar los XML Schemas en el lado del cliente.

Respecto a las técnicas de programación se puede decir que se usó en cierta medida el concepto de reutilización de código usando la herencia de Java, aunque todavía se pueden abstraer más las clases para hacer que el código en vez de ocupar 70 páginas utilice solamente 35.

Un punto que debemos destacar es que es bueno resolver un problema usando el paradigma o la forma de pensar conocida si se desea tener ventaja en tiempo, pero si se cuenta con tiempo para planear la elaboración de una aplicación siempre es bueno conocer todas las opciones y escoger cuál es la mejor para el problema en particular.

De la misma forma, antes de involucrarse en el proceso de desarrollo, siempre es bueno cerciorarse de que ya se cuenta con la última versión de las herramientas que se usarán Al 15 de febrero de 2004, las últimas versiones de las herramientas utilizadas fueron: Java 2 Standard Edition v.1.4.2; Java Web Services Developers Package v.1.3 (Windows i586); MySQL clients & servers v.4.0.17; MySQL ODBC v. 3.51.06 y MSSOAP Toolkit v.3.0. Si el lector desea desarrollar su propia aplicación de web services, debe actualizar éste “toolkit” o conjunto de herramientas para beneficiarse de las nuevas funcionalidades de las mismas.

La versión 2 del sistema puede instalarse en una PC solamente para propósitos de demonstración. La versión de la aplicación de la cual tomamos la idea se encuentra corriendo en la red de computadoras con enlace satelital propiedad de la industria automotriz en la cual laboré.

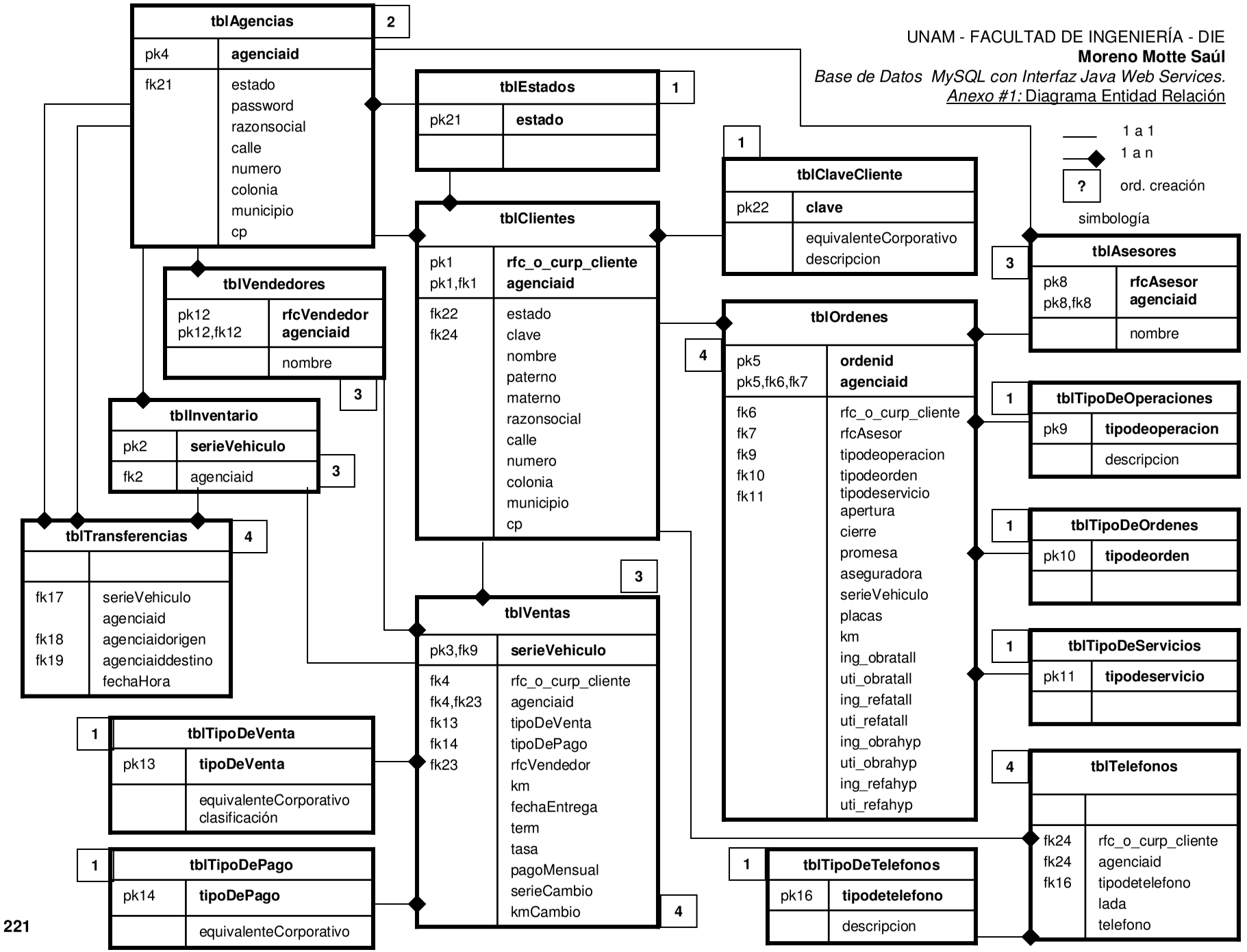
En éste punto no sobra invitar a los Ingenieros en Computación a adquirir un correcto dominio del idioma inglés, no conformarse con cumplir con ciertos tramites, porque los manuscritos e impresos de la bibliografía, así como las publicaciones de la ciencia de la computación se encuentran en dicho idioma. Tiene su mérito no solo traducir un texto, sino comprender y aplicar los conceptos comprendidos.

Además, la experiencia académica siempre es base de la experiencia profesional. En mi caso, sin la educación obtenida en la Facultad de Ingeniería, la tecnología que gira alrededor de las expresiones regulares hubiera pasado desapercibida. Sin las materias de Lenguajes Formales y Autómatas, o Compiladores, nunca hubiera conocido como se forma un AFD, ni hubiera visto la utilidad que reviste el que tecnología de punta construya dicho AFD de manera automática. Adicionalmente, sin la materia de bases de datos nunca hubiera podido diseñar una base de datos relacional normalizada, y sin la matería de Temas Selectos de Ingeniería en

Computación no hubiera aprendido a programar en Java ni conocer sus tecnologías asociadas. Todos estos conceptos fueron clave para la elaboración del proyecto sustentado en la tesis.

Finalmente, diré que la educación contínua es uno de los pilares de la mejora contínua. En los inicios de la computación, el lenguaje ensamblador era mejor que el lenguaje máquina; los intérpretes eran preferibles a los ensambladores; la programación estructurada llegó a ser mejor que los intérpretes y ahora, la programación orientada a objetos es el mejor camino para hacer sistemas de información. Es decir, así como el que creía que la programación estructurada sería el último avance de las ciencias de la computación se equivocó, hoy estaríamos equivocados si pensamos que la programación orientada a objetos será 'el último grito de la moda' en desarrollo de aplicaciones. ¿Cómo saber cuál será la forma de pensar y desarrollar aplicaciones el día de mañana? Se invita al lector, especialmente al estudiante de Ingeniería en Computación, aunque ello implique en ocasiones un pequeño desembolso económico, a capacitarse continuamente con cursos acerca de las nuevas herramientas de programación y de cómputo distribuido, o de su área de interés, cursos que ofrece la UNAM a través de la Facultad o el DGSCA, o bien cursos de certificación de marcas como Microsoft, Oracle o Sun Microsystems.

Apéndice


tblAgencias
pk4 agenciaid
fk21 estado, password, razonsocial, calle, numero, colonia, municipio, cp
2
tblEstados
pk21 estado
1
tblClaveCliente
pk22 clave
equivalenteCorporativo
descripcion
1
tblClientes
pk1 rfc_o_curp_cliente
pk1,fk1 agenciaid
fk22 estado
fk24 clave
nombre, paterno, materno, razonsocial, calle, numero, colonia, municipio, cp
tblVendedores
pk12 rfcVendedor
pk12,fk12 agenciaid
nombre
3
tblInventario
pk2 serieVehiculo
fk2 agenciaid
3
tblTransferencias
4
fk17 serieVehiculo
agenciaid
fk18 agenciaidorigen
fk19 agenciaiddestino
fechaHora
tblVentas
3
pk3,fk9 serieVehiculo
fk4 rfc_o_curp_cliente
fk4,fk23 agenciaid
fk13 tipoDeVenta
fk14 tipoDePago
fk23 rfcVendedor
km
fechaEntrega
term
tasa
pagoMensual
serieCambio
kmCambio
4
tblOrdenes
4
pk5 ordenid
pk5,fk6,fk7 agenciaid
fk6 rfc_o_curp_cliente
fk7 rfcAsesor
fk9 tipodeoperacion
fk10 tipodeorden
fk11 tipodeservicio
apertura
cierre
promesa
aseguradora
serieVehiculo
placas
km
ing_obratall
uti_obratall
ing_refatall
uti_refatall
ing_obrahyp
uti_obrahyp
ing_refahyp
uti_refahyp
tblAsesores
3
pk8 rfcAsesor
pk8,fk8 agenciaid
nombre
tblTipoDeOperaciones
1
pk9 tipodeoperacion
descripcion
tblTipoDeOrdenes
1
pk10 tipodeorden
tblTipoDeServicios
1
pk11 tipodeservicio
tblTelefonos
4
fk24 rfc_o_curp_cliente
fk24 agenciaid
fk16 tipodetelefono
lada
telefono
tblTipoDeVenta
1
pk13 tipoDeVenta
equivalenteCorporativo
clasificación
tblTipoDePago
1
pk14 tipoDePago
equivalenteCorporativo
tblTipoDeTelefonos
1
pk16 tipodetelefono
descripcion
1 a 1
1 a n
? ord. creación
simbología

Anexo 2. SQL para Definición de Estructuras de Datos en MySQL

produccion.sql

```
drop database if exists produccion;
create database produccion;
use produccion;

-- (1) CATALOGOS --
DROP TABLE IF EXISTS tblEstados;
CREATE TABLE tblEstados (
  estado char(20) NOT NULL default '',
  PRIMARY KEY  (estado)
) TYPE=InnoDB;
LOCK TABLES tblEstados WRITE;
INSERT INTO tblEstados VALUES ('AGS'), ('BC'), ('BCS'), ('CAMP'), ('CHIS'), ('CHIH'), ('COAH'),
('COL'), ('DF'), ('DGO'), ('MEX'), ('GTO'), ('GRO'), ('HGO'), ('JAL'), ('MICH'), ('MOR'), ('NAY'),
('NL'), ('OAX'), ('PUE'), ('QRO'), ('Q.ROO'), ('SLP'), ('SIN'), ('SON'), ('TAB'), ('TAMPS'), ('TLAX'),
('VER'), ('YUC'), ('ZAC');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDeOperaciones;
CREATE TABLE tblTipoDeOperaciones (
  tipodeoperacion char(20) NOT NULL default '',
  descripcion char(50),
  PRIMARY KEY  (tipodeoperacion)
) TYPE=InnoDB;
LOCK TABLES tblTipoDeOperaciones WRITE;
INSERT INTO tblTipoDeOperaciones VALUES ('mant','mantenimiento'),('repma','reparación mayor'),
('hyp','hojalatería y pintura');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDeOrdenes;
CREATE TABLE tblTipoDeOrdenes (
  tipodeorden char(20) NOT NULL default '',
  PRIMARY KEY  (tipodeorden)
) TYPE=InnoDB;
LOCK TABLES tblTipoDeOrdenes WRITE;
INSERT INTO tblTipoDeOrdenes VALUES ('publico'), ('interno'), ('garantia');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDeServicios;
CREATE TABLE tblTipoDeServicios (
  tipodeservicio char(20) NOT NULL default '',
  PRIMARY KEY  (tipodeservicio)
) TYPE=InnoDB;
LOCK TABLES tblTipoDeServicios WRITE;
INSERT INTO tblTipoDeServicios VALUES ('paquete'),('express');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDeVenta;
CREATE TABLE tblTipoDeVenta (
  tipodeventa char(20) NOT NULL default '',
  equivalenteCorporativo char (10),
  clasificacion char(100),
  PRIMARY KEY  (tipodeventa)
) TYPE=InnoDB;
LOCK TABLES tblTipoDeVenta WRITE;
```

```
INSERT INTO tblTipoDeVenta VALUES
('vmenudeo' ,'0','menudeo - retail'),
('vempleado','A','menudeo - empleados de contado'),
('vcomgere' ,'X','menudeo - compromisos gerenciales'),
('vnarrend' ,'6','menudeo - arrendamiento individual a través de arrendadora independiente'),
('varrend'  ,'2','menudeo - plan corporativo'),
('flotmenor','E','flotilla - compromisos gerenciales'),
('flotmayor','7','flotilla - flotilleros comerciales'),
('vtagobfed','S','flotilla - gobierno federal'),
('vtagobest','3','flotilla - gobierno estatal');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDePago;
CREATE TABLE tblTipoDePago (
  tipodepago char(20) NOT NULL default '',
  equivalenteCorporativo char(10),
  PRIMARY KEY  (tipodepago)
) TYPE=InnoDB;
LOCK TABLES tblTipoDePago WRITE;
INSERT INTO tblTipoDePago VALUES ('arrendamiento','L'),('financiamiento','F'),('contado','C');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblTipoDeTelefonos;
CREATE TABLE tblTipoDeTelefonos (
  tipodetelefono char(20) NOT NULL default '',
  descripcion char(50),
  PRIMARY KEY  (tipodetelefono)
) TYPE=InnoDB;
LOCK TABLES tblTipoDeTelefonos WRITE;
INSERT INTO tblTipoDeTelefonos VALUES
('compradorcasa','Telefono de Casa de Comprador'),
('compradoroficina','Telefono de Oficina de Comprador'),
('compradorcel','Telefono Celular de Comprador'),
('conductorcasa','Telefono de Casa de Conductor'),
('conductoroficina','Telefono de Oficina de Conductor'),
('conductorcel','Telefono Celular de Conductor');
UNLOCK TABLES;



DROP TABLE IF EXISTS tblClaveCliente;
CREATE TABLE tblClaveCliente (
  clave char(20) NOT NULL default '',
  equivalenteCorporativo char(10),
  descripcion char(50),
  PRIMARY KEY  (clave)
) TYPE=InnoDB;
LOCK TABLES tblClaveCliente WRITE;
INSERT INTO tblClaveCliente VALUES
('negocios','0','venta de negocios'),
('sr','1','señor'),
('sra','2','señora'),
('srta','3','señorita'),
('extranjero','8','venta al extranjero'),
('moral','9','venta a persona moral');
UNLOCK TABLES;

-- (2) tblAgencias --
DROP TABLE IF EXISTS tblAgencias;
CREATE TABLE tblAgencias (
  agenciaid char(20) NOT NULL default '',
  estado char(20) NOT NULL default '',
  password char(20),
  razonSocial char(100),
  calle char(50),
  numero char(50),
```

```
  colonia char(50),
  municipio char(50),
  cp char(10),
  PRIMARY KEY  (agenciaid),
  KEY estado (estado),
  FOREIGN KEY (estado) REFERENCES tblEstados(estado) ON UPDATE CASCADE
) TYPE=InnoDB;

-- (3) tblVendedores, tblInventario, tblClientes, tblAsesores --
DROP TABLE IF EXISTS tblVendedores;
CREATE TABLE tblVendedores (
  rfcVendedor char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  nombre char(50),
  PRIMARY KEY  (rfcVendedor, agenciaid),
  KEY agenciaid (agenciaid),
  FOREIGN KEY (agenciaid) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblAsesores;
CREATE TABLE tblAsesores (
  rfcAsesor char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  nombre char(50),
  PRIMARY KEY  (rfcAsesor, agenciaid),
  KEY agenciaid (agenciaid),
  FOREIGN KEY (agenciaid) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblInventario;
CREATE TABLE tblInventario (
  serieVehiculo char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  PRIMARY KEY  (serieVehiculo),
  KEY agenciaid (agenciaid),
  FOREIGN KEY (agenciaid) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblClientes;
CREATE TABLE tblClientes (
  rfc_o_curp_cliente char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  estado char(20) NOT NULL default '',
  clave char(20) NOT NULL default '',
  nombre char(100),
  paterno char(100),
  materno char(100),
  razonSocial char(100),
  calle char(50),
  numero char(50),
  colonia char(50),
  municipio char(50),
  cp char(10),
  PRIMARY KEY  (rfc_o_curp_cliente, agenciaid),
  KEY agenciaid (agenciaid),
  KEY estado (estado),
  KEY clave (clave),
  FOREIGN KEY (agenciaid) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE,
  FOREIGN KEY (estado) REFERENCES tblEstados(estado) ON UPDATE CASCADE,
  FOREIGN KEY (clave) REFERENCES tblClaveCliente(clave) ON UPDATE CASCADE
) TYPE=InnoDB;
```

```
-- (4) Maestros tblTransferencias, tblVentas, tblOrdenes & Catalogo tblTelefonos --
DROP TABLE IF EXISTS tblTransferencias;
CREATE TABLE tblTransferencias (
  serieVehiculo char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  agenciaidorigen char(20) NOT NULL default '',
  agenciaiddestino char(20) NOT NULL default '',
  fechahora datetime,
  KEY serieVehiculo (serieVehiculo),
  KEY agenciaidorigen (agenciaidorigen),
  KEY agenciaiddestino (agenciaiddestino),
  FOREIGN KEY (serieVehiculo) REFERENCES tblInventario(serieVehiculo) ON UPDATE CASCADE,
  FOREIGN KEY (agenciaidorigen) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE,
  FOREIGN KEY (agenciaiddestino) REFERENCES tblAgencias(agenciaid) ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblVentas;
CREATE TABLE tblVentas (
  serieVehiculo char(20) NOT NULL default '',
  rfc_o_curp_cliente char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  tipodeventa char(20) NOT NULL default '',
  tipodepago char(20) NOT NULL default '',
  rfcVendedor char(20) NOT NULL default '',
  km integer(10),
  fechaEntrega datetime,
  term integer(10),
  tasa decimal(12,2),
  pagoMensual decimal(12,2),
  serieCambio char(20),
  kmCambio integer(10),
  PRIMARY KEY (serieVehiculo),
  KEY serieVehiculo (serieVehiculo),
  KEY tipodeventa (tipodeventa),
  KEY tipodepago (tipodepago),
  KEY (serieVehiculo),
  INDEX (rfc_o_curp_cliente, agenciaid),
  INDEX (rfcVendedor, agenciaid),
  FOREIGN KEY (rfc_o_curp_cliente, agenciaid) REFERENCES tblClientes(rfc_o_curp_cliente, agenciaid)
  ON UPDATE CASCADE,
  FOREIGN KEY (rfcVendedor, agenciaid) REFERENCES tblVendedores(rfcVendedor, agenciaid)
  ON UPDATE CASCADE,
  FOREIGN KEY (serieVehiculo) REFERENCES tblInventario(serieVehiculo)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodeventa) REFERENCES tblTipoDeVenta(tipodeventa)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodepago) REFERENCES tblTipoDePago(tipodepago)
  ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblOrdenes;
CREATE TABLE tblOrdenes (
  ordenid char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  rfc_o_curp_cliente char(20) NOT NULL default '',
  rfcAsesor char(20) NOT NULL default '',
  tipodeoperacion char(20) NOT NULL default '',
  tipodeorden char(20) NOT NULL default '',
  tipodeservicio char(20) NOT NULL default '',
  apertura datetime,
  cierre datetime,
  promesa datetime,
  aseguradora char(100),
  serieVehiculo char(20),
  placas char(20),
  km integer(10),
```

```
  ing_obratall decimal(12,2),
  uti_obratall decimal(12,2),
  ing_refatall decimal(12,2),
  uti_refatall decimal(12,2),
  ing_obrahyp  decimal(12,2),
  uti_obrahyp  decimal(12,2),
  ing_refahyp  decimal(12,2),
  uti_refahyp  decimal(12,2),
  PRIMARY KEY (ordenid, agenciaid),
  KEY tipodeoperacion (tipodeoperacion),
  KEY tipodeorden (tipodeorden),
  KEY tipodeservicio (tipodeservicio),
  INDEX (rfc_o_curp_cliente, agenciaid),
  INDEX (rfcAsesor, agenciaid),
  FOREIGN KEY (rfc_o_curp_cliente, agenciaid) REFERENCES tblClientes(rfc_o_curp_cliente, agenciaid)
  ON UPDATE CASCADE,
  FOREIGN KEY (rfcAsesor, agenciaid) REFERENCES tblAsesores(rfcAsesor, agenciaid)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodeoperacion) REFERENCES tblTipoDeOperaciones(tipodeoperacion)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodeorden) REFERENCES tblTipoDeOrdenes(tipodeorden)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodeservicio) REFERENCES tblTipoDeServicios(tipodeservicio)
  ON UPDATE CASCADE
) TYPE=InnoDB;



DROP TABLE IF EXISTS tblTelefonos;
CREATE TABLE tblTelefonos (
  rfc_o_curp_cliente char(20) NOT NULL default '',
  agenciaid char(20) NOT NULL default '',
  tipodetelefono char(20) NOT NULL default '',
  lada char(20),
  telefono char(20),
  KEY tipodetelefono (tipodetelefono),
  INDEX (rfc_o_curp_cliente, agenciaid),
  FOREIGN KEY (rfc_o_curp_cliente, agenciaid) REFERENCES tblClientes(rfc_o_curp_cliente, agenciaid)
  ON UPDATE CASCADE,
  FOREIGN KEY (tipodetelefono) REFERENCES tblTipoDeTelefonos(tipodetelefono)
  ON UPDATE CASCADE
) TYPE=InnoDB;
```

Anexo 3. Utilería de Conexión JDBC para MySQL en entorno multiproceso o de uso concurrente

Índice de Programas

ConnectionPool.java

```
package coreservlets;

import java.sql.*;
import java.util.*;

/** A class for preallocating, recycling, and managing
 *  JDBC connections.
 *  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class ConnectionPool implements Runnable {
  private String driver, url, username, password, connString;
  private int maxConnections;
  private boolean waitIfBusy;
  private Vector availableConnections, busyConnections;
  private boolean connectionPending = false;

  public ConnectionPool(String driver, String url,
                        String username, String password,
                        int initialConnections,
                        int maxConnections,
                        boolean waitIfBusy)
      throws SQLException {
    this.driver = driver;
    this.url = url;
    this.username = username;
    this.password = password;
    this.maxConnections = maxConnections;
    this.waitIfBusy = waitIfBusy;
    if (initialConnections > maxConnections) {
      initialConnections = maxConnections;
    }
    availableConnections = new Vector(initialConnections);
    busyConnections = new Vector();
    for(int i=0; i<initialConnections; i++) {
      availableConnections.addElement(makeNewConnection());
    }
  }

  public ConnectionPool(String driver, String connString,
                        int initialConnections,
                        int maxConnections,
                        boolean waitIfBusy)
      throws SQLException {
    this.driver = driver;
    this.connString = connString;
    this.maxConnections = maxConnections;
    this.waitIfBusy = waitIfBusy;
    if (initialConnections > maxConnections) {
      initialConnections = maxConnections;
    }
    availableConnections = new Vector(initialConnections);
    busyConnections = new Vector();
    for(int i=0; i<initialConnections; i++) {
      availableConnections.addElement(makeNewConnection());
    }
  }
```

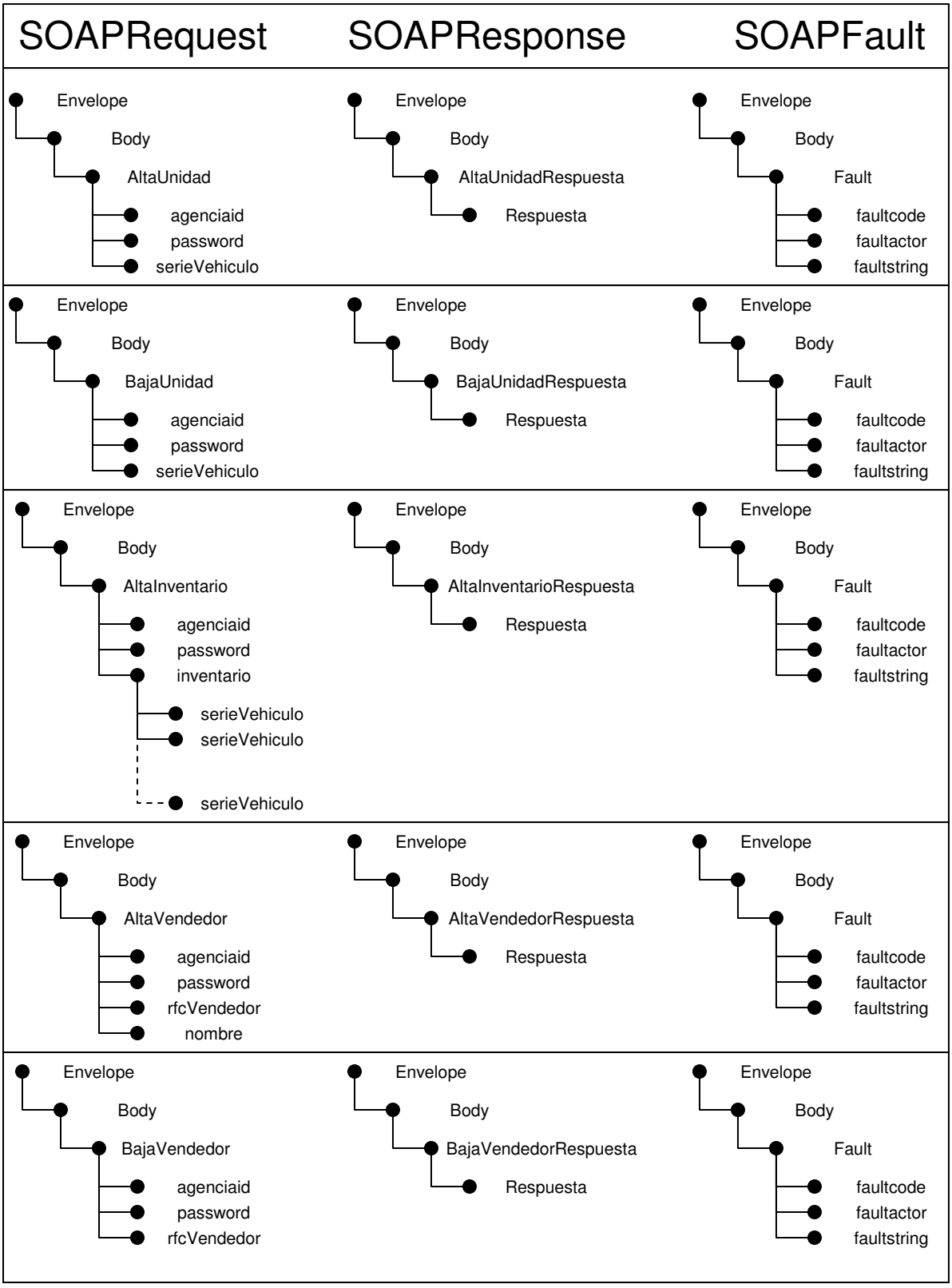
```
public synchronized Connection getConnection()
    throws SQLException {
  if (!availableConnections.isEmpty()) {
    Connection existingConnection =
      (Connection)availableConnections.lastElement();
    int lastIndex = availableConnections.size() - 1;
    availableConnections.removeElementAt(lastIndex);
    // If connection on available list is closed (e.g.,
    // it timed out), then remove it from available list
    // and repeat the process of obtaining a connection.
    // Also wake up threads that were waiting for a
    // connection because maxConnection limit was reached.
    if (existingConnection.isClosed()) {
      notifyAll(); // Freed up a spot for anybody waiting
      return(getConnection());
    } else {
      busyConnections.addElement(existingConnection);
      return(existingConnection);
    }
  } else {

    // Three possible cases:
    // 1) You haven't reached maxConnections limit. So
    //    establish one in the background if there isn't
    //    already one pending, then wait for
    //    the next available connection (whether or not
    //    it was the newly established one).
    // 2) You reached maxConnections limit and waitIfBusy
    //    flag is false. Throw SQLException in such a case.
    // 3) You reached maxConnections limit and waitIfBusy
    //    flag is true. Then do the same thing as in second
    //    part of step 1: wait for next available connection.

    if ((totalConnections() < maxConnections) &&
        !connectionPending) {
      makeBackgroundConnection();
    } else if (!waitIfBusy) {
      throw new SQLException("Connection limit reached");
    }
    // Wait for either a new connection to be established
    // (if you called makeBackgroundConnection) or for
    // an existing connection to be freed up.
    try {
      wait();
    } catch(InterruptedException ie) {}
    // Someone freed up a connection, so try again.
    return(getConnection());
  }
}

// You can't just make a new connection in the foreground
// when none are available, since this can take several
// seconds with a slow network connection. Instead,
// start a thread that establishes a new connection,
// then wait. You get woken up either when the new connection
// is established or if someone finishes with an existing
// connection.

private void makeBackgroundConnection() {
  connectionPending = true;
  try {
    Thread connectThread = new Thread(this);
    connectThread.start();
  } catch(OutOfMemoryError oome) {
    // Give up on new connection
  }
}
```

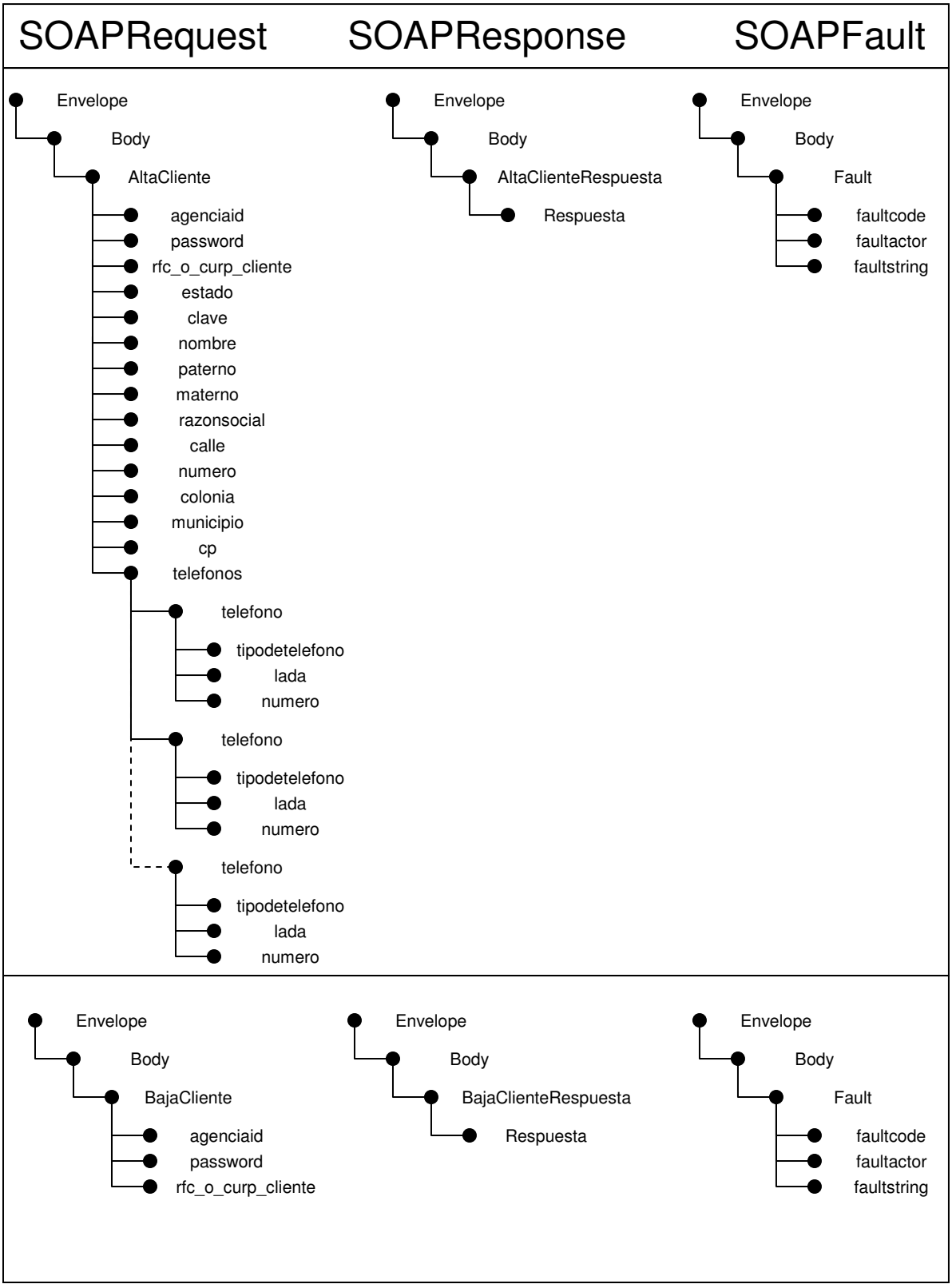
```
public void run() {
  try {
    Connection connection = makeNewConnection();
    synchronized(this) {
      availableConnections.addElement(connection);
      connectionPending = false;
      notifyAll();
    }
  } catch(Exception e) { // SQLException or OutOfMemory
    // Give up on new connection and wait for existing one
    // to free up.
  }
}

// This explicitly makes a new connection. Called in
// the foreground when initializing the ConnectionPool,
// and called in the background when running.

private Connection makeNewConnection()
    throws SQLException {
  try {
    // Load database driver if not already loaded
    Class.forName(driver).newInstance();
       //Class.forName("com.mysql.jdbc.Driver").newInstance();
    // Establish network connection to database
       //String connString =
       //     "jdbc:mysql://localhost/distribuidor?user=root&password=";
    Connection connection =
               DriverManager.getConnection(connString);
    //  DriverManager.getConnection(url, username, password);
       return(connection);
  } catch(ClassNotFoundException cnfe) {
    // Simplify try/catch blocks of people using this by
    // throwing only one exception type.
    throw new SQLException("ClassNotFoundException: " );
  } catch(InstantiationException ie) {
    throw new SQLException("InstantiationException: " );
     } catch(IllegalAccessException iae) {
    throw new SQLException("IllegalAccessException: " );
     }

}

public synchronized void free(Connection connection) {
  busyConnections.removeElement(connection);
  availableConnections.addElement(connection);
  // Wake up threads that are waiting for a connection
  notifyAll();
}

public synchronized int totalConnections() {
  return(availableConnections.size() +
         busyConnections.size());
}

/** Close all the connections. Use with caution:
 *  be sure no connections are in use before
 *  calling. Note that you are not <I>required</I> to
 *  call this when done with a ConnectionPool, since
 *  connections are guaranteed to be closed when
 *  garbage collected. But this method gives more control
 *  regarding when the connections are closed.
 */

public synchronized void closeAllConnections() {
  closeConnections(availableConnections);
  availableConnections = new Vector();
  closeConnections(busyConnections);
  busyConnections = new Vector();
```

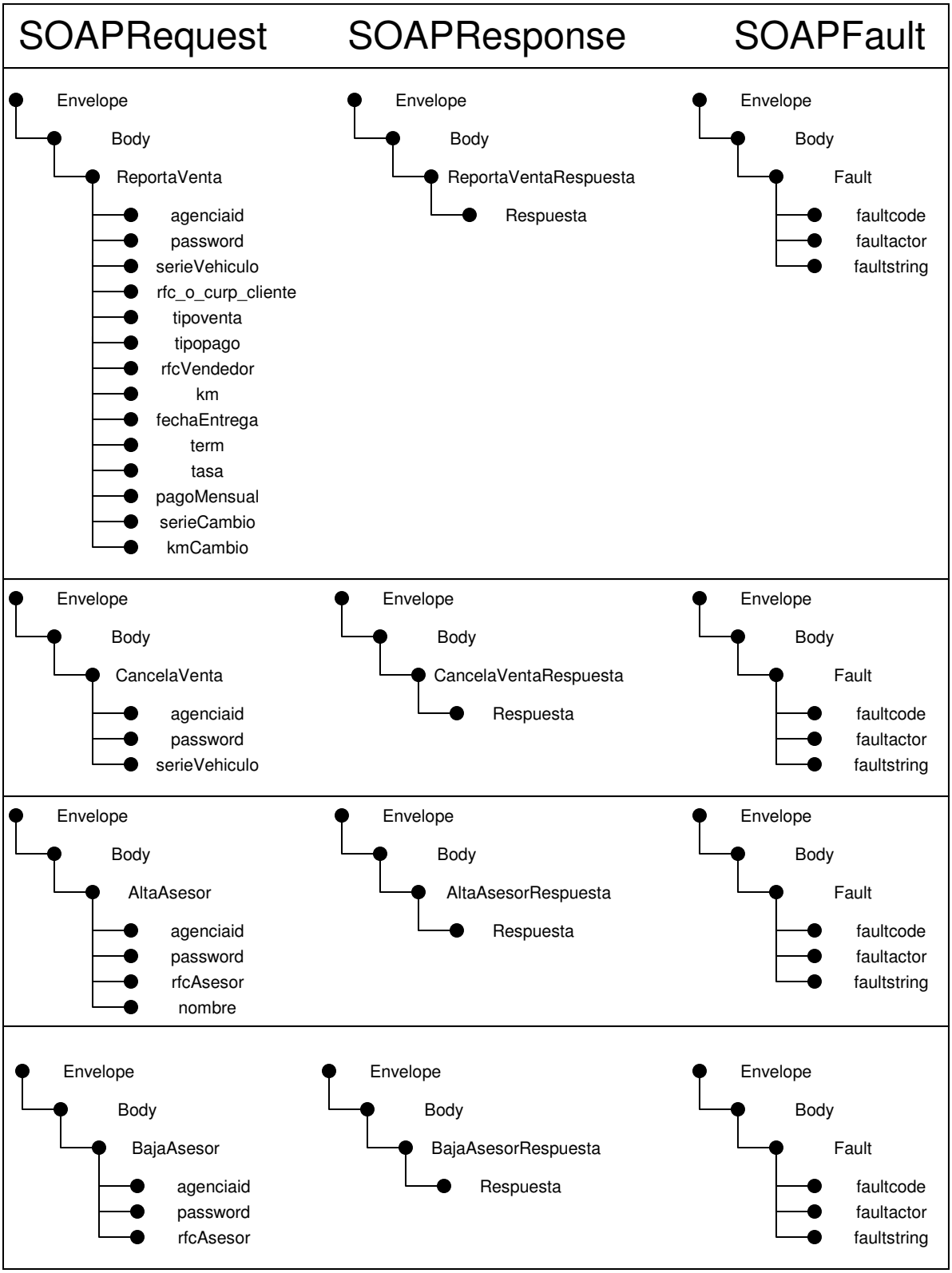
```
  }

  private void closeConnections(Vector connections) {
    try {
      for(int i=0; i<connections.size(); i++) {
        Connection connection =
          (Connection)connections.elementAt(i);
        if (!connection.isClosed()) {
          connection.close();
        }
      }
    } catch(SQLException sqle) {
      // Ignore errors; garbage collect anyhow
    }
  }

  public synchronized String toString() {
    String info =
      "ConnectionPool(" + url + "," + username + ")" +
      ", available=" + availableConnections.size() +
      ", busy=" + busyConnections.size() +
      ", max=" + maxConnections;
    return(info);
  }
}
```

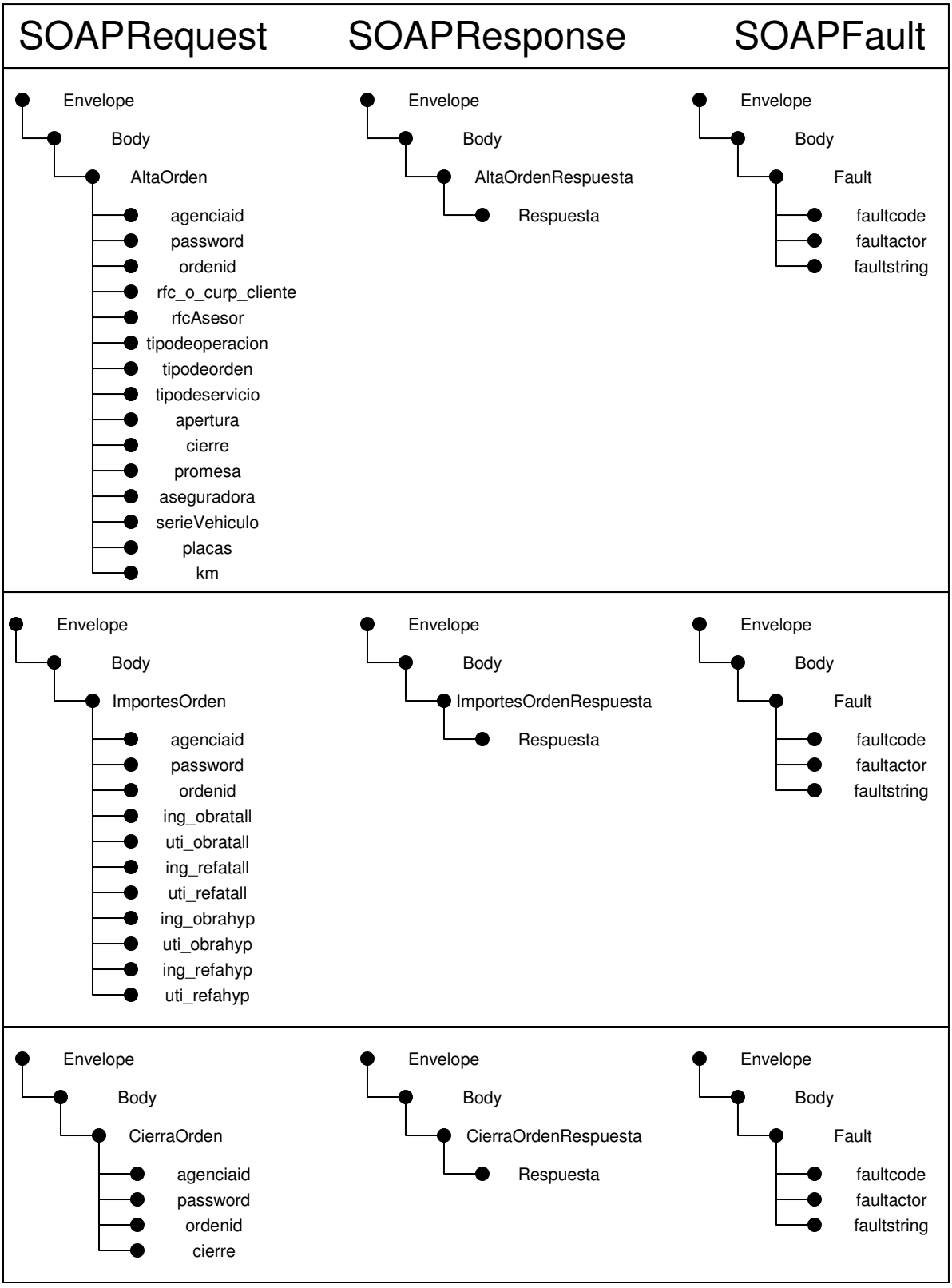
DriverUtilities.java

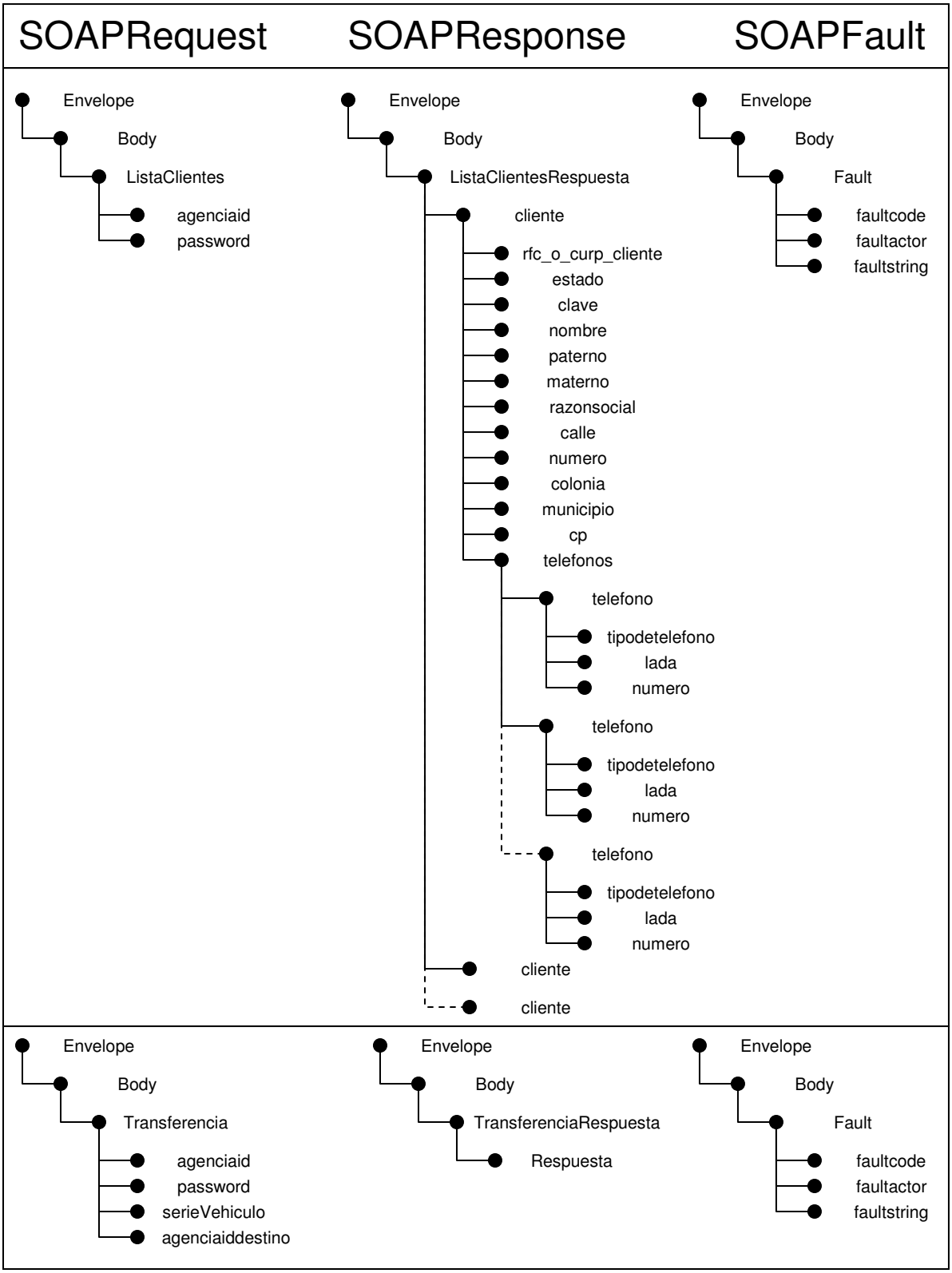
```
package coreservlets;

/** Some simple utilities for building Oracle and Sybase
 *  JDBC connections. This is <I>not</I> general-purpose
 *  code -- it is specific to my local setup.
 *  <P>
 *  Taken from Core Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.coreservlets.com/.
 *  &copy; 2000 Marty Hall; may be freely used or adapted.
 */

public class DriverUtilities {
  public static final int ORACLE = 1;
  public static final int SYBASE = 2;
  public static final int UNKNOWN = -1;

  /** Build a URL in the format needed by the
   *  Oracle and Sybase drivers I am using.
   */

  public static String makeURL(String host, String dbName,
                               int vendor) {
    if (vendor == ORACLE) {
      return("jdbc:oracle:thin:@" + host + ":1521:" + dbName);
    } else if (vendor == SYBASE) {
      return("jdbc:sybase:Tds:" + host  + ":1521" +
             "?SERVICENAME=" + dbName);
    } else {
      return(null);
    }
  }

  /** Get the fully qualified name of a driver. */

  public static String getDriver(int vendor) {
    if (vendor == ORACLE) {
      return("oracle.jdbc.driver.OracleDriver");
    } else if (vendor == SYBASE) {
      return("com.sybase.jdbc.SybDriver");
    } else {
      return(null);
    }
  }

  /** Map name to int value. */

  public static int getVendor(String vendorName) {
    if (vendorName.equalsIgnoreCase("oracle")) {
      return(ORACLE);
    } else if (vendorName.equalsIgnoreCase("sybase")) {
      return(SYBASE);
    } else {
      return(UNKNOWN);
    }
  }
}
```

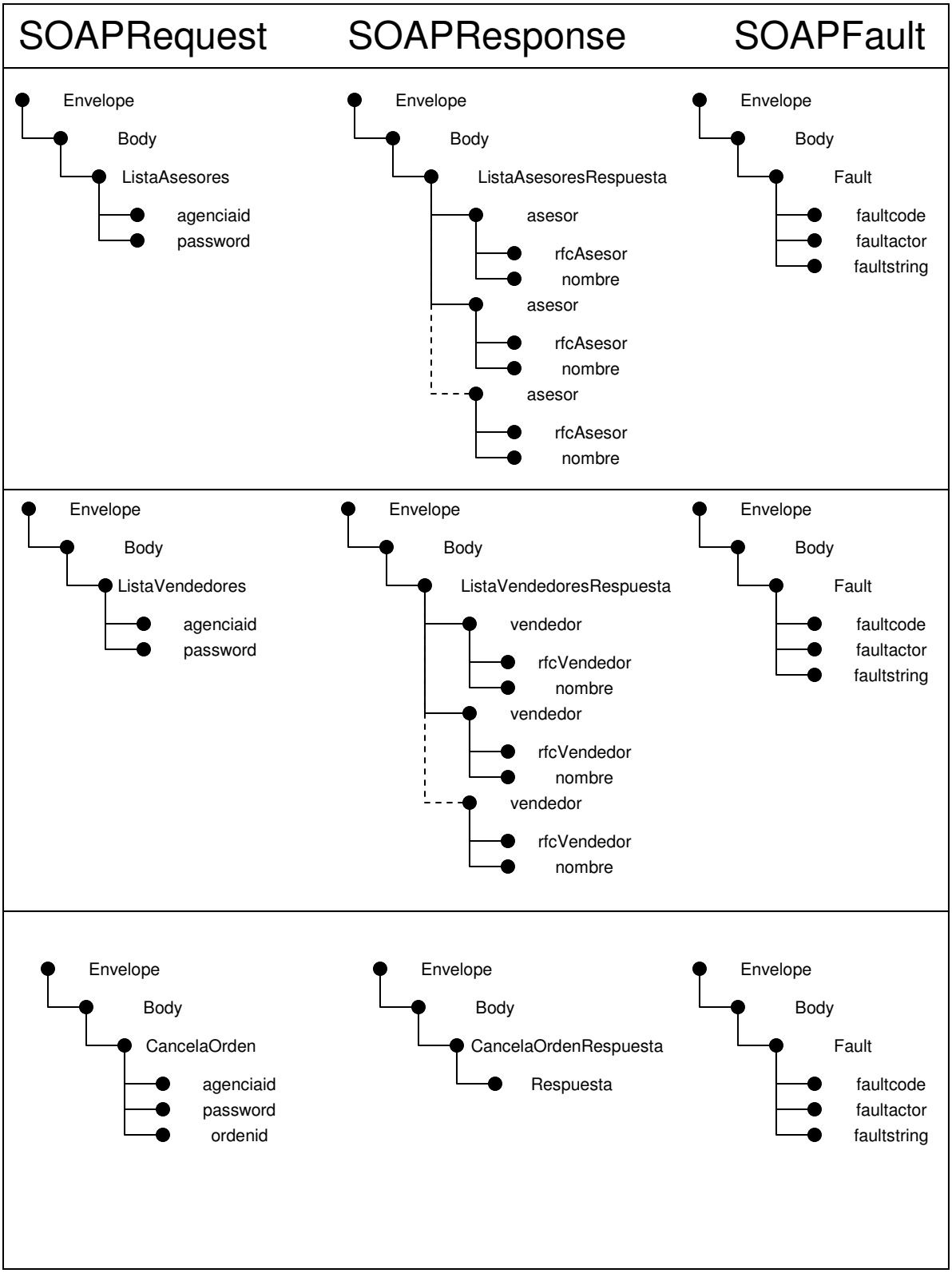

SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
AltaUnidad
agenciaid
password
serieVehiculo
Envelope
Body
AltaUnidadRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
BajaUnidad
agenciaid
password
serieVehiculo
Envelope
Body
BajaUnidadRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
AltaInventario
agenciaid
password
inventario
serieVehiculo
serieVehiculo
serieVehiculo
Envelope
Body
AltaInventarioRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
AltaVendedor
agenciaid
password
rfcVendedor
nombre
Envelope
Body
AltaVendedorRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
BajaVendedor
agenciaid
password
rfcVendedor
Envelope
Body
BajaVendedorRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring


SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
AltaCliente
agenciaid
password
rfc_o_curp_cliente
estado
clave
nombre
paterno
materno
razonsocial
calle
numero
colonia
municipio
cp
telefonos
telefono
tipodetelefono
lada
numero
telefono
tipodetelefono
lada
numero
telefono
tipodetelefono
lada
numero
Envelope
Body
AltaClienteRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
BajaCliente
agenciaid
password
rfc_o_curp_cliente
Envelope
Body
BajaClienteRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring


SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
ReportaVenta
agenciaid
password
serieVehiculo
rfc_o_curp_cliente
tipoventa
tipopago
rfcVendedor
km
fechaEntrega
term
tasa
pagoMensual
serieCambio
kmCambio
Envelope
Body
ReportaVentaRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
CancelaVenta
agenciaid
password
serieVehiculo
Envelope
Body
CancelaVentaRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
AltaAsesor
agenciaid
password
rfcAsesor
nombre
Envelope
Body
AltaAsesorRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
BajaAsesor
agenciaid
password
rfcAsesor
Envelope
Body
BajaAsesorRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring


SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
AltaOrden
agenciaid
password
ordenid
rfc_o_curp_cliente
rfcAsesor
tipodeoperacion
tipodeorden
tipodeservicio
apertura
cierre
promesa
aseguradora
serieVehiculo
placas
km
Envelope
Body
AltaOrdenRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
ImportesOrden
agenciaid
password
ordenid
ing_obratall
uti_obratall
ing_refatall
uti_refatall
ing_obrahyp
uti_obrahyp
ing_refahyp
uti_refahyp
Envelope
Body
ImportesOrdenRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
CierraOrden
agenciaid
password
ordenid
cierre
Envelope
Body
CierraOrdenRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring


SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
ListaClientes
agenciaid
password
Envelope
Body
ListaClientesRespuesta
cliente
rfc_o_curp_cliente
estado
clave
nombre
paterno
materno
razonsocial
calle
numero
colonia
municipio
cp
telefonos
telefono
tipodetelefono
lada
numero
telefono
tipodetelefono
lada
numero
telefono
tipodetelefono
lada
numero
cliente
cliente
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
Transferencia
agenciaid
password
serieVehiculo
agenciaiddestino
Envelope
Body
TransferenciaRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring


SOAPRequest
SOAPResponse
SOAPFault
Envelope
Body
ListaAsesores
agenciaid
password
Envelope
Body
ListaAsesoresRespuesta
asesor
rfcAsesor
nombre
asesor
rfcAsesor
nombre
asesor
rfcAsesor
nombre
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
ListaVendedores
agenciaid
password
Envelope
Body
ListaVendedoresRespuesta
vendedor
rfcVendedor
nombre
vendedor
rfcVendedor
nombre
vendedor
rfcVendedor
nombre
Envelope
Body
Fault
faultcode
faultactor
faultstring
Envelope
Body
CancelaOrden
agenciaid
password
ordenid
Envelope
Body
CancelaOrdenRespuesta
Respuesta
Envelope
Body
Fault
faultcode
faultactor
faultstring

Anexo 5. Sistema Java Web Services Versión 02

Índice de Programas

AltaAsesor.java

```
1    // Bibliotecas para validar el mensaje XML SOAP
2    import org.iso_relax.verifier.*;
3    import com.sun.msv.driver.textui.ReportErrorHandler;
4    import org.xml.sax.*;
5
6    // Bibliotecas para procesar el mensaje XML SOAP
7    import java.io.*;
8    import javax.xml.soap.*;
9    import java.util.*;
10
11   // Bibliotecas para manejar la conexión de sql
12   import java.sql.*;
13   import coreservlets.*;
14
15   public class AltaAsesor
16   {
17            private String agenciaid="";
18            private String password="";
19            private String rfcAsesor="";
20            private String nombre="";
21            private boolean estatus = false;
22            private String codigoError = "5010-No hay datos para procesar";
23       static MessageFactory fac = null;
24
25       static {
26           try {
27               fac = MessageFactory.newInstance();
28           } catch (Exception ex) {
29               ex.printStackTrace();
30           }
31       }
32
33            public AltaAsesor(MimeHeaders headers, InputStream is)     throws IOException {
34                    try
35                    {
36                            SOAPMessage msg = fac.createMessage(headers, is);
37               SOAPPart part = msg.getSOAPPart();
38
39                            // Código Agregado para la versión 2.0 del sistema
40                            // para validar datos desde la llegada del xml
41                            VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
42                                                                                                                    Schema schema =
                                      factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaAsesor.xsd");
43                            Verifier verifier = schema.newVerifier();
44                            verifier.verify(part);
45                            // fin de código para version 2.0
46
47               SOAPEnvelope envelope = part.getEnvelope();
48               SOAPBody body = envelope.getBody();
49                            java.util.Iterator iterator = body.getChildElements();
50                            SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
51                            iterator = addUnit.getChildElements();
52                            SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
53                            SOAPBodyElement password = (SOAPBodyElement)iterator.next();
54                            SOAPBodyElement rfcAsesor = (SOAPBodyElement)iterator.next();
55                            SOAPBodyElement nombre = (SOAPBodyElement)iterator.next();
56                            this.agenciaid=agenciaid.getValue();
57                            this.password=password.getValue();
58                            this.rfcAsesor =rfcAsesor.getValue();
59                            this.nombre =nombre.getValue();
60                    }
61                    catch (SOAPException e)
62                    {
63                            this.estatus=false;
64                            this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
65                    }
66                    catch (VerifierConfigurationException e) // catch requerido para version 2.0
67                    {
68                            this.estatus=false;
69                            this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
70                    }
71                    catch (SAXException e) // catch requerido para version 2.0
72                    {
73                            this.estatus=false;
74                            this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
```

```
75                  }
76
77          }
78
79          public AltaAsesor(String agenciaid, String password, String rfcAsesor, String nombre) {
80                  this.agenciaid=agenciaid;
81                  this.password=password;
82                  this.rfcAsesor=rfcAsesor;
83                  this.nombre=nombre;
84          }
85
86          public boolean ejecuta(ConnectionPool connectionPool) {
87                  String strSQL;
88                  // if requerido para version 2.0
89                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
90                  {
91                  try {
92
93                          Connection miconexion = connectionPool.getConnection();
94                          String comilla ="'";
95                          ResultSet rs ,rs2;
96                          String serieVehiculo;
97                          PreparedStatement pstmInsert;
98                          Statement stm = miconexion.createStatement();
99                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
100                                  "WHERE agenciaid = " + comilla + agenciaid + comilla +
101                                      "AND password = password("+comilla + password + comilla +")";
102                         rs = stm.executeQuery(strSQL);
103                         rs.next();
104                         if (rs.getInt("conteo")!=1)
105                         {
106                                 this.estatus=false;
107                             this.codigoError="5000 - agenciaid:" + agenciaid +
108                                         " con password:" + password + " no son validos. Contacte al administrador ";
109                         } else {
110                                 this.estatus=true;
111                                 strSQL = "SELECT count(*) as conteo FROM tblAsesores " +
112                          " WHERE rfcAsesor = " + comilla + this.rfcAsesor + comilla +
113                          " AND agenciaid = " + comilla + this.agenciaid + comilla ;
114                                 rs = stm.executeQuery(strSQL);
115                                 rs.next();
116                                 if (rs.getInt("conteo")==1) {
117                                         this.estatus=false;
118                                     this.codigoError="5430 - El asesor " + this.rfcAsesor +
119                                                 " ya fue dado de alta previamente para la agencia " + this.agenciaid
                                                                                                                 ;
120                                 } else {
121                                         this.estatus=true;
122                                         strSQL = "INSERT INTO tblAsesores(rfcAsesor,agenciaid,nombre) VALUES "+
123                                                         "(" + comilla + this.rfcAsesor + comilla +
124                                                         "," + comilla + this.agenciaid + comilla +
125                                                         "," + comilla + this.nombre + comilla +")";
126                                         pstmInsert = miconexion.prepareStatement(strSQL);
127                                         pstmInsert.execute();
128                                         pstmInsert.close();
129                                 }// end if
130                         }
131                         rs.close();
132                         miconexion.close();
133                 } catch(SQLException sqle) {
134                   switch (sqle.getErrorCode())
135                   {
136                   case 0:
137                     this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
138                          break;
139                   case 1049:
140                     this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
141                          break;
142                   case 1216: //para fks multiples se tendra que revisar que datos no son validos
143                        this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                        administrador ";
144                          break;
145                   default:
146                            this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                    Contacte al administrador ";
147                   }// end switch
148                   //connectionPool = null;
149                   this.estatus=false;
150                 }//end catch
```

```
151                 } //end if
152                 return (this.estatus);
153           }// end ejecuta
154
155
156           public SOAPMessage respuesta() {
157                   if (this.estatus)
158                   {
159                           return mensajeExitoso();
160                   } else {
161                           return mensajeError();
162                   }// end if
163           }
164
165           public SOAPMessage mensajeExitoso() {
166       SOAPMessage message = null;
167
168           try {
169               // create price list message
170               message = fac.createMessage();
171
172               // Access the SOAPBody object
173               SOAPPart part = message.getSOAPPart();
174               SOAPEnvelope envelope = part.getEnvelope();
175               SOAPBody body = envelope.getBody();
176                           SOAPHeader header = envelope.getHeader();
177                           header.detachNode();
178
179               Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
180                   "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
181               SOAPBodyElement list = body.addBodyElement(bodyName);
182
183               // coffee
184               Name coffeeN = envelope.createName("Respuesta");
185               SOAPElement coffee = list.addChildElement(coffeeN);
186                           coffee.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
187               message.saveChanges();
188
189           } catch(Exception e) {
190               e.printStackTrace();
191           }
192           return message;
193       }// end mensajeExitoso
194
195           public SOAPMessage mensajeError() {
196       SOAPMessage message = null;
197
198           try {
199               // create price list message
200               message = fac.createMessage();
201
202
203               // Access the SOAPBody object
204               SOAPPart part = message.getSOAPPart();
205               SOAPEnvelope envelope = part.getEnvelope();
206               SOAPBody body = envelope.getBody();
207                           SOAPHeader header = envelope.getHeader();
208                           header.detachNode();
209
210                           SOAPFactory soapFactory = SOAPFactory.newInstance();
211                           SOAPFault fault = body.addFault();
212                           Name faultName =
213                                   soapFactory.createName("Server",
214                                   "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
215                           fault.setFaultCode(faultName);
216                             fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                    clase
217                           fault.setFaultString(this.codigoError);
218               message.saveChanges();
219
220           } catch(Exception e) {
221               e.printStackTrace();
222           }
223           return message;
224       }// end servicioError
225
226
227           public String getAgenciaId() {
228                   return this.agenciaid;
```

```
229             }
230
231             public String getPassword() {
232                     return this.password;
233             }
234
235             public boolean getEstatus() {
236                     return this.estatus;
237             }
238
239             public String getCodigoError(){
240                     return this.codigoError;
241             }
242
243             public void setCodigoError(String codigoError) {
244                     this.codigoError=codigoError;
245             }
246
247     }
```

AltaCliente.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class AltaCliente
16      {
17              private String agenciaid="";
18              private String password="";
19              private String rfc_o_curp_cliente="";
20              private String estado="";
21              private String clave="";
22              private String nombre="";
23              private String paterno="";
24              private String materno="";
25              private String razonSocial="";
26              private String calle="";
27              private String numeroCalle="";
28              private String colonia="";
29              private String municipio="";
30              private String cp="";
31              private Vector telefonos= new Vector();
32              private int cantidadTelefonos=0;
33              Telefono telefono;
34              private boolean estatus = false;
35              private String codigoError = "5010-No hay datos para procesar";
36          static MessageFactory fac = null;
37
38          static {
39              try {
40                  fac = MessageFactory.newInstance();
41              } catch (Exception ex) {
42                  ex.printStackTrace();
43              }
44          }
45
46              public AltaCliente(MimeHeaders headers, InputStream is)    throws IOException {
47                      try
48                      {
49                              SOAPMessage msg = fac.createMessage(headers, is);
50                  SOAPPart part = msg.getSOAPPart();
51
52                              // Código Agregado para la versión 2.0 del sistema
53                              // para validar datos desde la llegada del xml
54                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
55                                                                                              Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaCliente.xsd");
```

```
56                              Verifier verifier = schema.newVerifier();
57                              verifier.verify(part);
58                              // fin de código para version 2.0
59
60                              SOAPEnvelope envelope = part.getEnvelope();
61                      SOAPBody body = envelope.getBody();
62                              java.util.Iterator iterator = body.getChildElements();
63                              SOAPBodyElement altacliente = (SOAPBodyElement)iterator.next();
64                              iterator = altacliente.getChildElements();
65                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
66                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
67                              SOAPBodyElement rfc_o_curp_cliente = (SOAPBodyElement)iterator.next();
68                              SOAPBodyElement estado = (SOAPBodyElement)iterator.next();
69                              SOAPBodyElement clave = (SOAPBodyElement)iterator.next();
70                              SOAPBodyElement nombre = (SOAPBodyElement)iterator.next();
71                              SOAPBodyElement paterno = (SOAPBodyElement)iterator.next();
72                              SOAPBodyElement materno = (SOAPBodyElement)iterator.next();
73                              SOAPBodyElement razonSocial = (SOAPBodyElement)iterator.next();
74                              SOAPBodyElement calle = (SOAPBodyElement)iterator.next();
75                              SOAPBodyElement numeroCalle = (SOAPBodyElement)iterator.next();
76                              SOAPBodyElement colonia = (SOAPBodyElement)iterator.next();
77                              SOAPBodyElement municipio = (SOAPBodyElement)iterator.next();
78                              SOAPBodyElement cp = (SOAPBodyElement)iterator.next();
79                              SOAPBodyElement telefonosSOAP = (SOAPBodyElement)iterator.next();
80                              this.agenciaid=agenciaid.getValue();
81                              this.password=password.getValue();
82                              this.rfc_o_curp_cliente =rfc_o_curp_cliente.getValue();
83                              this.estado=estado.getValue();
84                              this.clave = clave.getValue();
85                              this.nombre= nombre.getValue();
86                              this.paterno=paterno.getValue();
87                              this.materno=materno.getValue();
88                              this.razonSocial=razonSocial.getValue();
89                              this.calle=calle.getValue();
90                              this.numeroCalle=numeroCalle.getValue();
91                              this.colonia=colonia.getValue();
92                              this.municipio=municipio.getValue();
93                              this.cp=cp.getValue();
94                              java.util.Iterator iteratorTelefonos = telefonosSOAP.getChildElements();
95                              while (iteratorTelefonos.hasNext())
96                              {
97                                      SOAPBodyElement telefonoSOAP=(SOAPBodyElement)iteratorTelefonos.next();
98                                      java.util.Iterator iteratorTelefono =telefonoSOAP.getChildElements();
99                                      SOAPBodyElement tipodetelefono= (SOAPBodyElement)iteratorTelefono.next();
100                                     SOAPBodyElement lada= (SOAPBodyElement)iteratorTelefono.next();
101                                     SOAPBodyElement numeroTelefono= (SOAPBodyElement)iteratorTelefono.next();
102                                                                                                     telefono=new
                                                Telefono(tipodetelefono.getValue(),lada.getValue(),numeroTelefono.getValue());
103                                     //escribe el vector de objetos Telefono
104                                     telefonos.addElement(telefono);
105                                     cantidadTelefonos=cantidadTelefonos+1;
106                             }
107                     }
108                     catch (SOAPException e)
109                     {
110                             this.estatus=false;
111                             this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
112                     }
113                     catch (VerifierConfigurationException e) // catch requerido para version 2.0
114                     {
115                             this.estatus=false;
116                             this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
117                     }
118                     catch (SAXException e) // catch requerido para version 2.0
119                     {
120                             this.estatus=false;
121                             this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
122                     }
123
124             }
125
126             public AltaCliente(String agenciaid, String password, Vector telefonos) {
127                     this.agenciaid=agenciaid;
128                     this.password=password;
129                     this.telefonos=telefonos; //es un vector de objetos tipo Telefono (ver Telefono.java)
130             }
131
132             public boolean ejecuta(ConnectionPool connectionPool) {
133                     String strSQL;
```

```
134                     // if requerido para version 2.0
135                     if (this.codigoError.equals( "5010-No hay datos para procesar"))
136                     {
137                     try {
138                             Connection miconexion = connectionPool.getConnection();
139                             miconexion.setAutoCommit(false);
140                             String comilla ="'";
141                             ResultSet rs ,rs2;
142                             //String serieVehiculo;
143                             PreparedStatement pstmInsert;
144                             Statement stm = miconexion.createStatement();
145                             strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
146                                      "WHERE agenciaid = " + comilla + agenciaid + comilla +
147                                           "AND password = password("+comilla + password + comilla +")";
148                             rs = stm.executeQuery(strSQL);
149                             rs.next();
150                             if (rs.getInt("conteo")!=1) //if 1
151                             {
152                                     this.estatus=false;
153                                 this.codigoError="5000 - agenciaid:" + agenciaid +
154                                             " con password:" + password + " no son validos. Contacte al administrador ";
155                             } else {
156                                     strSQL = "SELECT count(*) as conteo FROM tblEstados " +
157                                      "WHERE estado = " + comilla + this.estado + comilla ;
158                                     rs = stm.executeQuery(strSQL);
159                                     rs.next();
160                                     if (rs.getInt("conteo")==0) // if 2
161                                     {
162                                             this.estatus=false;
163                                             this.codigoError="5030 - estado:" + this.estado+ " no es valido. ";
164                                     } else {
165                                             strSQL = "SELECT count(*) as conteo FROM tblClaveCliente " +
166                                              " WHERE clave = " + comilla + this.clave + comilla ;
167                                             rs = stm.executeQuery(strSQL);
168                                             rs.next();
169                                             if (rs.getInt("conteo")==0) // if 3
170                                             {
171                                                     this.estatus=false;
172                                                            this.codigoError="5040 - clave cliente:" + this.clave+ " no es
                                                                                                                    valida. ";
173                                             } else {
174                                                     this.estatus=true;
175                                                     strSQL= "SELECT count(*) as conteo FROM tblClientes " +
176                                                                                     " WHERE rfc_o_curp_cliente = " + comilla +
                                                                                             this.rfc_o_curp_cliente + comilla +
177                                                                             " AND agenciaid = " + comilla + this.agenciaid +
                                                                                                                    comilla;
178                                                     rs=stm.executeQuery(strSQL);
179                                                     rs.next();
180                                                     if (rs.getInt("conteo")==1) { //if 4
181                                                             this.estatus=false;
182                                                                     this.codigoError="5040 - El cliente:" +
                                                                                             this.rfc_o_curp_cliente+
183                                                                     " ya esta dado de alta en la agencia " +
                                                                                             this.agenciaid;
184                                                     } else if (cantidadTelefonos==0)  {
185                                                             this.estatus=false;
186                                                                     this.codigoError="5050 - Los datos del cliente se han
                                                                                             enviado sin telefonos";
187                                                     } else {
188                                                             this.estatus=true;
189                                                                             strSQL="INSERT INTO tblClientes
                                                                                     (rfc_o_curp_cliente,agenciaid," +
190                                                                             "estado,clave, nombre, paterno, materno,
                                                                                             razonsocial, " +
191                                                                     "calle, numero,colonia, municipio,cp)  VALUES " +
192                                                                      "(" + comilla + this.rfc_o_curp_cliente + comilla
                                                                                                                    +
193                                                                     "," + comilla + this.agenciaid + comilla +
194                                                                     "," + comilla + this.estado  + comilla +
195                                                                     "," + comilla + this.clave   + comilla +
196                                                                     "," + comilla + this.nombre  + comilla +
197                                                                     "," + comilla + this.paterno + comilla +
198                                                                     "," + comilla + this.materno + comilla +
199                                                                     "," + comilla + this.razonSocial + comilla +
200                                                                     "," + comilla + this.calle + comilla +
201                                                                     "," + comilla + this.numeroCalle + comilla +
202                                                                     "," + comilla + this.colonia + comilla +
203                                                                     "," + comilla + this.municipio + comilla +
```

```
204                                                                                    "," + comilla + this.cp + comilla + ")";
205                                                                          pstmInsert = miconexion.prepareStatement(strSQL);
206                                                                          pstmInsert.execute();
207                                                                          for (int i=0;i< cantidadTelefonos ;i++ ) {
208                                                                                  telefono = (Telefono)telefonos.elementAt(i);
209                                                                                          strSQL = "SELECT count(*) as conteo FROM
                                                                                                                                  tblTelefonos " +
210                                                                                                  " WHERE rfc_o_curp_cliente = " + comilla +
                                                                                                                this.rfc_o_curp_cliente + comilla +
211                                                                                      " AND agenciaid = " + comilla + this.agenciaid +
                                                                                                                                      comilla +
212                                                                                                                  " AND tipodetelefono = "+
                                                                                      comilla+telefono.getTipoDeTelefono() + comilla +
213                                                                                       " AND lada = " + comilla + telefono.getLada() +
                                                                                                                                      comilla+
214                                                                                    " AND telefono = " + comilla + telefono.getNumero()
                                                                                                                                    + comilla;
215                                                                                  rs = stm.executeQuery(strSQL);
216                                                                                  rs.next();
217                                                                                  if (rs.getInt("conteo")==0) { //if 5
218                                                                                          strSQL = "SELECT count(*) as conteo FROM
                                                                                                                                  tblTipoDeTelefonos " +
219                                                                                                                  " WHERE tipodetelefono= "
                                                                                        +comilla+telefono.getTipoDeTelefono()+comilla;
220                                                                                                  rs = stm.executeQuery(strSQL);
221                                                                                                  rs.next();
222                                                                                                    if (rs.getInt("conteo")==0) { //if
                                                                                                                                          6
223                                                                                                          this.estatus=false;
224                                                                                                          this.codigoError="5060 - El
                                                                                                                  tipo de telefono" +
225
        telefono.getTipoDeTelefono()+ " no es valido";
226                                                                                                          miconexion.rollback();
227                                                                                                          break;
228                                                                                                  } else {
229                                                                                                           strSQL = "INSERT INTO
                                                                                                   tblTelefonos(rfc_o_curp_cliente," +
230
                                                                                      "agenciaid,tipodetelefono,lada,telefono) VALUES "+
231                                                                                                                          "(" + comilla +
                                                                                                   this.rfc_o_curp_cliente + comilla +
232                                                                                                                          "," + comilla +
                                                                                                     this.agenciaid + comilla +
233                                                                                                                          "," + comilla +
                                                                                                telefono.getTipoDeTelefono() + comilla +
234                                                                                                                          "," + comilla +
                                                                                                     telefono.getLada() + comilla +
235                                                                                                                          "," + comilla +
                                                                                                  telefono.getNumero() + comilla +")";
236                                                                                                                            pstmInsert =
                                                                                                  miconexion.prepareStatement(strSQL);
237                                                                                                           pstmInsert.execute();
238                                                                                                   } // end if 6
239                                                                                  }// end if 5
240                                                                          } // next i
241                                                                  }//end if 4
242                                                          } // end if 3
243                                                  } // end if 2
244                                          }// end if 1
245                                          rs.close();
246                                          miconexion.commit();
247                                          miconexion.close();
248                                  } catch(SQLException sqle) {
249                                    switch (sqle.getErrorCode())
250                                    {
251                                    case 0:
252                                               //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                                  "+Integer.toString(inventario.size());
253                                          break;
254                                    case 1049:
255                                      this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
256                                          break;
257                                    case 1216: //para fks multiples se tendra que revisar que datos no son validos
258                                      this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al administrador
                                                                                                                                          ";
259                                          break;
260                                    default:
261                                             this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
```

```
                                                                    Contacte al administrador ";
                    } // end switch
                    //connectionPool = null;
                    this.estatus=false;
                } //end catch
                } //end if
                return (this.estatus);
        }


        public SOAPMessage respuesta() {
                if (this.estatus)
                {
                        return mensajeExitoso();
                } else {
                        return mensajeError();
                }// end if
        }

        public SOAPMessage mensajeExitoso() {
    SOAPMessage message = null;

        try {
            // create price list message
            message = fac.createMessage();

            // Access the SOAPBody object
            SOAPPart part = message.getSOAPPart();
            SOAPEnvelope envelope = part.getEnvelope();
            SOAPBody body = envelope.getBody();
                        SOAPHeader header = envelope.getHeader();
                        header.detachNode();

            Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
                "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
            SOAPBodyElement list = body.addBodyElement(bodyName);

            Name exitosoN = envelope.createName("Respuesta");
            SOAPElement exitoso = list.addChildElement(exitosoN);
                        exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
            message.saveChanges();

        } catch(Exception e) {
            e.printStackTrace();
        }
        return message;
    }// end mensajeExitoso

        public SOAPMessage mensajeError() {
    SOAPMessage message = null;

        try {
            // create price list message
            message = fac.createMessage();


            // Access the SOAPBody object
            SOAPPart part = message.getSOAPPart();
            SOAPEnvelope envelope = part.getEnvelope();
            SOAPBody body = envelope.getBody();
                        SOAPHeader header = envelope.getHeader();
                        header.detachNode();

                        SOAPFactory soapFactory = SOAPFactory.newInstance();
                        SOAPFault fault = body.addFault();
                        Name faultName =
                                soapFactory.createName("Server",
                                "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
                        fault.setFaultCode(faultName);
                          fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
                        fault.setFaultString(this.codigoError);
            message.saveChanges();

        } catch(Exception e) {
            e.printStackTrace();
        }
        return message;
    }// end servicioError
```

```
339
340
341             public String getAgenciaId() {
342                     return this.agenciaid;
343             }
344
345             public String getPassword() {
346                     return this.password;
347             }
348
349             public boolean getEstatus() {
350                     return this.estatus;
351             }
352
353             public String getCodigoError(){
354                     return this.codigoError;
355             }
356
357             public void setCodigoError(String codigoError) {
358                     this.codigoError=codigoError;
359             }
360
361     }
```

AltaInventario.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class AltaInventario
16      {
17              private String agenciaid="";
18              private String password="";
19              private Vector inventario= new Vector();
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22          static MessageFactory fac = null;
23
24          static {
25              try {
26                  fac = MessageFactory.newInstance();
27              } catch (Exception ex) {
28                  ex.printStackTrace();
29              }
30          }
31
32              public AltaInventario(MimeHeaders headers, InputStream is) throws IOException {
33                      try
34                      {
35                              SOAPMessage msg = fac.createMessage(headers, is);
36                  SOAPPart part = msg.getSOAPPart();
37
38                              // Código Agregado para la versión 2.0 del sistema
39                              // para validar datos desde la llegada del xml
40                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaInventario.xsd");
42                              Verifier verifier = schema.newVerifier();
43                              verifier.verify(part);
44                              // fin de código para version 2.0
45
46                  SOAPEnvelope envelope = part.getEnvelope();
47                  SOAPBody body = envelope.getBody();
48                              java.util.Iterator iterator = body.getChildElements();
49                              SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                              iterator = addUnit.getChildElements();
51                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
```

```
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement inventarioSOAP = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                          iterator = inventarioSOAP.getChildElements();
57                          while (iterator.hasNext())
58                          {
59                                  inventario.addElement(
60                                          //escribe el vector de objetos string
61                                          ((SOAPBodyElement)iterator.next()).getValue()
62                                  );
63                          }
64                  }
65                  catch (SOAPException e)
66                  {
67                          this.estatus=false;
68                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
69                  }
70                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
71                  {
72                          this.estatus=false;
73                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
74                  }
75                  catch (SAXException e) // catch requerido para version 2.0
76                  {
77                          this.estatus=false;
78                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
79                  }
80
81          }
82
83          public AltaInventario(String agenciaid, String password, Vector inventario) {
84                  this.agenciaid=agenciaid;
85                  this.password=password;
86                  this.inventario=inventario; //es un vector de objetos String
87          }
88
89          public boolean ejecuta(ConnectionPool connectionPool) {
90                  String strSQL;
91                  // if requerido para version 2.0
92                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
93                  {
94
95                  try {
96
97                          Connection miconexion = connectionPool.getConnection();
98                          String comilla ="'";
99                          ResultSet rs ,rs2;
100                         String serieVehiculo;
101                         PreparedStatement pstmInsert;
102                         Statement stm = miconexion.createStatement();
103                         strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
104                                  "WHERE agenciaid = " + comilla + agenciaid + comilla +
105                                       "AND password = password("+comilla + password + comilla +")";
106                         rs = stm.executeQuery(strSQL);
107                         rs.next();
108                         if (rs.getInt("conteo")!=1)
109                         {
110                                 this.estatus=false;
111                             this.codigoError="5000 - agenciaid:" + agenciaid +
112                                         " con password:" + password + " no son validos. Contacte al administrador ";
113                         } else {
114                                 this.estatus=true;
115                                 for (int i=0;i< inventario.size() ;i++ ) {
116                                         serieVehiculo = (String)inventario.elementAt(i);
117                                         strSQL = "SELECT count(*) as conteo FROM tblInventario " +
118                                          " WHERE serieVehiculo = " + comilla + serieVehiculo + comilla ;
119                                             // " AND agenciaid = " + comilla + this.agenciaid + comilla ;
120                                         rs = stm.executeQuery(strSQL);
121                                         rs.next();
122                                         if (rs.getInt("conteo")==0) {
123                                                   strSQL = "INSERT INTO tblInventario(serieVehiculo,agenciaid) VALUES
                                                                                                                        "+
124                                                                  "(" + comilla + serieVehiculo + comilla +
125                                                                  "," + comilla + this.agenciaid + comilla +")";
126                                                 pstmInsert = miconexion.prepareStatement(strSQL);
127                                                 pstmInsert.execute();
128                                                 pstmInsert.close();
129                                         }// end if
```

```
130                                 } // next i
131                             }
132                             rs.close();
133                             miconexion.close();
134                     } catch(SQLException sqle) {
135                       switch (sqle.getErrorCode())
136                       {
137                       case 0:
138                         this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
139                             break;
140                       case 1049:
141                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
142                             break;
143                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
144                            this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                    administrador ";
145                             break;
146                       default:
147                                this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                        Contacte al administrador ";
148                       } //end switch
149                       //connectionPool = null;
150                       this.estatus=false;
151                     }//end catch
152                     }//end if
153                     return (this.estatus);
154             }
155
156
157             public SOAPMessage respuesta() {
158                     if (this.estatus)
159                     {
160                             return mensajeExitoso();
161                     } else {
162                             return mensajeError();
163                     }// end if
164             }
165
166             public SOAPMessage mensajeExitoso() {
167         SOAPMessage message = null;
168
169             try {
170                 // create price list message
171                 message = fac.createMessage();
172
173                 // Access the SOAPBody object
174                 SOAPPart part = message.getSOAPPart();
175                 SOAPEnvelope envelope = part.getEnvelope();
176                 SOAPBody body = envelope.getBody();
177                             SOAPHeader header = envelope.getHeader();
178                             header.detachNode();
179
180                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
181                     "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
182                 SOAPBodyElement list = body.addBodyElement(bodyName);
183
184                 // coffee
185                 Name coffeeN = envelope.createName("Respuesta");
186                 SOAPElement coffee = list.addChildElement(coffeeN);
187                             coffee.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
188                 message.saveChanges();
189
190             } catch(Exception e) {
191                 e.printStackTrace();
192             }
193             return message;
194         }// end mensajeExitoso
195
196             public SOAPMessage mensajeError() {
197         SOAPMessage message = null;
198
199             try {
200                 // create price list message
201                 message = fac.createMessage();
202
203
204                 // Access the SOAPBody object
205                 SOAPPart part = message.getSOAPPart();
206                 SOAPEnvelope envelope = part.getEnvelope();
```

```
207                 SOAPBody body = envelope.getBody();
208                             SOAPHeader header = envelope.getHeader();
209                             header.detachNode();
210
211                             SOAPFactory soapFactory = SOAPFactory.newInstance();
212                             SOAPFault fault = body.addFault();
213                             Name faultName =
214                                     soapFactory.createName("Server",
215                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
216                             fault.setFaultCode(faultName);
217                               fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
218                             fault.setFaultString(this.codigoError);
219                 message.saveChanges();
220
221             } catch(Exception e) {
222                 e.printStackTrace();
223             }
224             return message;
225         }// end servicioError
226
227
228             public String getAgenciaId() {
229                     return this.agenciaid;
230             }
231
232             public String getPassword() {
233                     return this.password;
234             }
235
236             public boolean getEstatus() {
237                     return this.estatus;
238             }
239
240             public String getCodigoError(){
241                     return this.codigoError;
242             }
243
244             public void setCodigoError(String codigoError) {
245                     this.codigoError=codigoError;
246             }
247
248     }
```

AltaOrden.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class AltaOrden
16      {
17              private String agenciaid="";
18              private String password="";
19              private String ordenid="";
20              private String rfc_o_curp_cliente="";
21              private String rfcAsesor="";
22              private String tipodeoperacion="";
23              private String tipodeorden="";
24              private String tipodeservicio="";
25              private String apertura="";
26              private String cierre="";
27              private String promesa="";
28              private String aseguradora="";
29              private String serieVehiculo="";
30              private String placas="";
```

```
31              private String km="";
32              private boolean estatus = false;
33              private String codigoError = "5010-No hay datos para procesar";
34          static MessageFactory fac = null;
35
36          static {
37              try {
38                  fac = MessageFactory.newInstance();
39              } catch (Exception ex) {
40                  ex.printStackTrace();
41              }
42          }
43
44              public AltaOrden(MimeHeaders headers, InputStream is)      throws IOException {
45                      try
46                      {
47                              SOAPMessage msg = fac.createMessage(headers, is);
48                      SOAPPart part = msg.getSOAPPart();
49
50                              // Código Agregado para la versión 2.0 del sistema
51                              // para validar datos desde la llegada del xml
52                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
53                                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaOrden.xsd");
54                              Verifier verifier = schema.newVerifier();
55                              verifier.verify(part);
56                              // fin de código para version 2.0
57
58                              SOAPEnvelope envelope = part.getEnvelope();
59                              SOAPBody body = envelope.getBody();
60                              java.util.Iterator iterator = body.getChildElements();
61                              SOAPBodyElement altaorden = (SOAPBodyElement)iterator.next();
62                              iterator = altaorden.getChildElements();
63                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
64                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
65                              SOAPBodyElement ordenid = (SOAPBodyElement)iterator.next();
66                              SOAPBodyElement rfc_o_curp_cliente = (SOAPBodyElement)iterator.next();
67                              SOAPBodyElement rfcAsesor = (SOAPBodyElement)iterator.next();
68                              SOAPBodyElement tipodeoperacion = (SOAPBodyElement)iterator.next();
69                              SOAPBodyElement tipodeorden = (SOAPBodyElement)iterator.next();
70                              SOAPBodyElement tipodeservicio = (SOAPBodyElement)iterator.next();
71                              SOAPBodyElement apertura = (SOAPBodyElement)iterator.next();
72                              SOAPBodyElement cierre = (SOAPBodyElement)iterator.next();
73                              SOAPBodyElement promesa = (SOAPBodyElement)iterator.next();
74                              SOAPBodyElement aseguradora = (SOAPBodyElement)iterator.next();
75                              SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
76                              SOAPBodyElement placas = (SOAPBodyElement)iterator.next();
77                              SOAPBodyElement km = (SOAPBodyElement)iterator.next();
78                              this.agenciaid=agenciaid.getValue();
79                              this.password=password.getValue();
80                              this.ordenid = ordenid.getValue();
81                              this.rfc_o_curp_cliente=rfc_o_curp_cliente.getValue();
82                              this.rfcAsesor = rfcAsesor.getValue();
83                              this.tipodeoperacion= tipodeoperacion.getValue();
84                              this.tipodeorden= tipodeorden.getValue();
85                              this.tipodeservicio= tipodeservicio.getValue();
86                              this.apertura=apertura.getValue();
87                              this.cierre=cierre.getValue();
88                              this.promesa=promesa.getValue();
89                              this.aseguradora=aseguradora.getValue();
90                              this.serieVehiculo=serieVehiculo.getValue();
91                              this.placas=placas.getValue();
92                              this.km=km.getValue();
93                      }
94                      catch (SOAPException e)
95                      {
96                              this.estatus=false;
97                              this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
98                      }
99                      catch (VerifierConfigurationException e) // catch requerido para version 2.0
100                     {
101                             this.estatus=false;
102                             this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
103                     }
104                     catch (SAXException e) // catch requerido para version 2.0
105                     {
106                             this.estatus=false;
107                             this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
108                     }
```

```
109
110             }
111
112             public AltaOrden(String agenciaid, String password, String ordenid,
113                     String rfc_o_curp_cliente, String rfcAsesor, String tipodeoperacion,
114                     String tipodeorden, String tipodeservicio) {
115                     this.agenciaid=agenciaid;
116                     this.password=password;
117                     this.ordenid=ordenid;
118                     this.rfc_o_curp_cliente=rfc_o_curp_cliente;
119                     this.rfcAsesor = rfcAsesor;
120                     this.tipodeoperacion=tipodeoperacion;
121                     this.tipodeorden=tipodeorden;
122                     this.tipodeservicio=tipodeservicio;
123             }
124
125             public boolean ejecuta(ConnectionPool connectionPool) {
126                     String strSQL;
127                     // if requerido para version 2.0
128                     if (this.codigoError.equals( "5010-No hay datos para procesar"))
129                     {
130                     try {
131                             Connection miconexion = connectionPool.getConnection();
132                             miconexion.setAutoCommit(false);
133                             String comilla ="'";
134                             ResultSet rs ,rs2;
135                             //String serieVehiculo;
136                             PreparedStatement pstm;
137                             Statement stm = miconexion.createStatement();
138                             strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
139                                      " WHERE agenciaid = " + comilla + agenciaid + comilla +
140                                           " AND password = password("+comilla + password + comilla +")";
141                             rs = stm.executeQuery(strSQL);
142                             rs.next();
143                             if (rs.getInt("conteo")!=1) //if 1
144                             {
145                                     this.estatus=false;
146                                 this.codigoError="5000 - agenciaid:" + agenciaid +
147                                                 " con password:" + password + " no son validos. Contacte al administrador ";
148                             } else {
149                                     strSQL = "SELECT count(*) as conteo FROM tblClientes" +
150                                      " WHERE rfc_o_curp_cliente = " + comilla + this.rfc_o_curp_cliente + comilla +
151                                               " AND agenciaid = " + comilla + this.agenciaid + comilla;
152                                     rs = stm.executeQuery(strSQL);
153                                     rs.next();
154                                     if (rs.getInt("conteo")==0) {// if 2
155                                             this.estatus=false;
156                                             this.codigoError="5300 - El cliente " + this.rfc_o_curp_cliente +
157                                                     " no ha sido dado de alta para la agencia " + this.agenciaid;
158                                     } else {
159                                             strSQL = "SELECT count(*) as conteo FROM tblAsesores " +
160                                              " WHERE rfcAsesor = " + comilla + this.rfcAsesor + comilla +
161                                                       " AND agenciaid = " + comilla + this.agenciaid + comilla;
162                                             rs = stm.executeQuery(strSQL);
163                                             rs.next();
164                                             if (rs.getInt("conteo")==0) // if 3
165                                             {
166                                                     this.estatus=false;
167                                                     this.codigoError="5310 - El asesor " + this.rfcAsesor +
168                                                             " no ha sido dado de alta para la agencia " +
    this.agenciaid;
169                                             } else {
170                                                     strSQL= "SELECT count(*) as conteo FROM tblTipoDeOperaciones " +
171                                                                  " WHERE tipodeoperacion = " + comilla +
    this.tipodeoperacion + comilla;
172                                                     rs=stm.executeQuery(strSQL);
173                                                     rs.next();
174                                                     if (rs.getInt("conteo")==0) { //if 4
175                                                             this.estatus=false;
176                                                             this.codigoError="5320 - El tipodeoperacion " +
    this.tipodeoperacion +
177                                                                     " no es valido ";
178                                                     } else {
179                                                             strSQL = "SELECT count(*) as conteo FROM tblTipoDeOrdenes "
    +
180                                                              " WHERE tipodeorden = " + comilla + this.tipodeorden + comilla;
181                                                             rs=stm.executeQuery(strSQL);
182                                                             rs.next();
183                                                             if (rs.getInt("conteo")==0) { //if 5
```

```
184                                                     this.estatus=false;
185                                                             this.codigoError="5320 - El tipodeorden " +
                                                                                      this.tipodeorden +
186                                                     " no es valido ";
187                                                 } else {
188                                                         strSQL = "SELECT count(*) as conteo FROM
                                                                                  tblTipoDeServicios " +
189                                                       " WHERE tipodeservicio = " + comilla +
                                                                        this.tipodeservicio + comilla;
190                                                     rs=stm.executeQuery(strSQL);
191                                                     rs.next();
192                                                     if (rs.getInt("conteo")==0) { //if 6
193                                                             this.estatus=false;
194                                                              this.codigoError="5330 - El tipodeservicio
                                                                                  " + this.tipodeservicio +
195                                                             " no es valido ";
196                                                     } else {
197                                                         strSQL = "SELECT count(*) as conteo FROM
                                                                                        tblOrdenes " +
198                                                                 " WHERE ordenid = " + comilla +
                                                                                this.ordenid + comilla +
199                                                                 " AND agenciaid = " + comilla +
                                                                                this.agenciaid + comilla;
200                                                             rs=stm.executeQuery(strSQL);
201                                                             rs.next();
202                                                             if (rs.getInt("conteo")==1) { //if 7
203                                                                     this.estatus=false;
204                                                                       this.codigoError="5340 - La orden
                                                                                          " + this.ordenid +
205                                                                       " ya ha sido dado de alta para la
                                                                                agencia "+this.agenciaid;
206                                                             } else {
207                                                                     this.estatus=true;
208                                                                         strSQL="INSERT INTO tblOrdenes
                                                                                    (ordenid,agenciaid," +
209                                                                       "rfc_o_curp_cliente,rfcAsesor,
                                                                        tipodeoperacion, tipodeorden,"+
210                                                                       "tipodeservicio, apertura,cierre,
                                                                                                      "+
211                                                                                "promesa,aseguradora,
                                                                    serieVehiculo,placas,km)  VALUES " +
212                                                                "(" + comilla + this.ordenid + comilla +
213                                                                "," + comilla + this.agenciaid + comilla
                                                                                                      +
214                                                                 "," + comilla + this.rfc_o_curp_cliente
                                                                                          + comilla +
215                                                                      "," + comilla + this.rfcAsesor   +
                                                                                          comilla +
216                                                                 "," + comilla + this.tipodeoperacion  +
                                                                                          comilla +
217                                                                      "," + comilla + this.tipodeorden +
                                                                                          comilla +
218                                                                   "," + comilla + this.tipodeservicio +
                                                                                          comilla +
219                                                                 "," + comilla + this.apertura + comilla
                                                                                                      +
220                                                                "," + comilla + this.cierre + comilla +
221                                                                "," + comilla + this.promesa + comilla +
222                                                                      "," + comilla + this.aseguradora +
                                                                                          comilla +
223                                                                   "," + comilla + this.serieVehiculo +
                                                                                          comilla +
224                                                                "," + comilla + this.placas + comilla +
225                                                                "," + this.km +       ")";
226                                                             pstm = miconexion.prepareStatement(strSQL);
227                                                                      pstm.execute();
228                                                             } // end if #7
229                                                     }//end if #6
230                                                 }//end if #5
231                                             }//end if 4
232                                         } // end if 3
233                                     } // end if 2
234                             }// end if 1
235                             rs.close();
236                             miconexion.commit();
237                             miconexion.close();
238                     } catch(SQLException sqle) {
239                       switch (sqle.getErrorCode())
240                       {
```

```
241                     case 0:
242                               //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                    "+Integer.toString(inventario.size());
243                           break;
244                     case 1049:
245                       this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
246                           break;
247                     case 1216: //para fks multiples se tendra que revisar que datos no son validos
248                                   this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                                  administrador ";
249                           break;
250                     default:
251                             this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                    Contacte al administrador ";
252                     } // end switch
253                     //connectionPool = null;
254                     this.estatus=false;
255                   } // end catch
256                   } // end if
257                   return (this.estatus);
258           }
259
260
261           public SOAPMessage respuesta() {
262                   if (this.estatus)
263                   {
264                           return mensajeExitoso();
265                   } else {
266                           return mensajeError();
267                   }// end if
268           }
269
270           public SOAPMessage mensajeExitoso() {
271       SOAPMessage message = null;
272
273           try {
274               // create price list message
275               message = fac.createMessage();
276
277               // Access the SOAPBody object
278               SOAPPart part = message.getSOAPPart();
279               SOAPEnvelope envelope = part.getEnvelope();
280               SOAPBody body = envelope.getBody();
281                           SOAPHeader header = envelope.getHeader();
282                           header.detachNode();
283
284               Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
285                   "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
286               SOAPBodyElement list = body.addBodyElement(bodyName);
287
288               Name exitosoN = envelope.createName("Respuesta");
289               SOAPElement exitoso = list.addChildElement(exitosoN);
290                           exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
291               message.saveChanges();
292
293           } catch(Exception e) {
294               e.printStackTrace();
295           }
296           return message;
297       }// end mensajeExitoso
298
299           public SOAPMessage mensajeError() {
300       SOAPMessage message = null;
301
302           try {
303               // create price list message
304               message = fac.createMessage();
305
306
307               // Access the SOAPBody object
308               SOAPPart part = message.getSOAPPart();
309               SOAPEnvelope envelope = part.getEnvelope();
310               SOAPBody body = envelope.getBody();
311                           SOAPHeader header = envelope.getHeader();
312                           header.detachNode();
313
314                           SOAPFactory soapFactory = SOAPFactory.newInstance();
315                           SOAPFault fault = body.addFault();
316                           Name faultName =
```

```
317                                     soapFactory.createName("Server",
318                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
319                                 fault.setFaultCode(faultName);
320                                  fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
321                                 fault.setFaultString(this.codigoError);
322                 message.saveChanges();
323
324             } catch(Exception e) {
325                 e.printStackTrace();
326             }
327             return message;
328         }// end servicioError
329
330
331             public String getAgenciaId() {
332                     return this.agenciaid;
333             }
334
335             public String getPassword() {
336                     return this.password;
337             }
338
339             public boolean getEstatus() {
340                     return this.estatus;
341             }
342
343             public String getCodigoError(){
344                     return this.codigoError;
345             }
346
347             public void setCodigoError(String codigoError) {
348                     this.codigoError=codigoError;
349             }
350
351     }
```

AltaUnidad.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class AltaUnidad
16      {
17              private String agenciaid="";
18              private String password="";
19              private String serieVehiculo="";
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22          static MessageFactory fac = null;
23
24          static {
25              try {
26                  fac = MessageFactory.newInstance();
27              } catch (Exception ex) {
28                  ex.printStackTrace();
29              }
30          }
31
32              public AltaUnidad(MimeHeaders headers, InputStream is)     throws IOException {
33                      try
34                      {
35                              SOAPMessage msg = fac.createMessage(headers, is);
36                  SOAPPart part = msg.getSOAPPart();
37
```

```
38                          // Código Agregado para la versión 2.0 del sistema
39                          // para validar datos desde la llegada del xml
40                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                  Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaUnidad.xsd");
42                          Verifier verifier = schema.newVerifier();
43                          verifier.verify(part);
44                          // fin de código para version 2.0
45
46              SOAPEnvelope envelope = part.getEnvelope();
47              SOAPBody body = envelope.getBody();
48                          java.util.Iterator iterator = body.getChildElements();
49                          SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                          iterator = addUnit.getChildElements();
51                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                          this.serieVehiculo = serieVehiculo.getValue();
57                  }
58                  catch (SOAPException e)
59                  {
60                          this.estatus=false;
61                          this.codigoError="6000-Error al leer el xml de entrada en AltaUnidad";
62                  }
63                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                  {
65                          this.estatus=false;
66                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                  }
68                  catch (SAXException e) // catch requerido para version 2.0
69                  {
70                          this.estatus=false;
71                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                  }
73
74          }
75
76          public AltaUnidad(String agenciaid, String password, String serieVehiculo) {
77                  this.agenciaid=agenciaid;
78                  this.password=password;
79                  this.serieVehiculo=serieVehiculo;
80          }
81
82          public boolean ejecuta(ConnectionPool connectionPool) {
83                  String strSQL;
84                  // if requerido para version 2.0
85                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
86                  {
87                          try {
88                          Connection miconexion = connectionPool.getConnection();
89                          String comilla ="'";
90                          Statement stm = miconexion.createStatement();
91                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
92                                   "WHERE agenciaid = " + comilla + agenciaid + comilla +
93                                       "AND password = password("+comilla + password + comilla +")";
94                          ResultSet rs = stm.executeQuery(strSQL);
95                          rs.next();
96                          if (rs.getInt("conteo")!=1)
97                          {
98                                  this.estatus=false;
99                              this.codigoError="5000 - agenciaid:" + agenciaid +
100                                         " con password:" + password + " no son validos. Contacte al administrador ";
101                         } else {
102                                 strSQL = "INSERT INTO tblInventario VALUES (" +
103                                           comilla + this.serieVehiculo + comilla + "," +
104                                               comilla + this.agenciaid + comilla + ")";
105                                 PreparedStatement pstmInsert = miconexion.prepareStatement(strSQL);
106                                 pstmInsert.execute();
107                                 pstmInsert.close();
108                                 this.estatus=true;
109                         }
110                         rs.close();
111                         miconexion.close();
112                         } catch(SQLException sqle) {
113                           switch (sqle.getErrorCode())
114                           {
115                           case 0:
```

```
116                                         this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al
                                                                                                        administrador ";
117                                 break;
118                             case 1049:
119                                 this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
120                                 break;
121                             case 1062:
122                                         this.codigoError="1062 - Unidad " + this.serieVehiculo + " ya se encuentra en
                                                                                                        Inventario ";
123                                 break;
124                             case 1216: //para fks multiples se tendra que revisar que datos no son validos
125                                   this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo.
                                                                                        Contacte al administrador ";
126                                 break;
127                             default:
128                                  this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de
                                                                    datos. Contacte al administrador ";
129                             } //end switch
130                             //connectionPool = null;
131                             this.estatus=false;
132                         } //end catch
133                 } //end if
134                 return (this.estatus);
135         }
136
137         public SOAPMessage respuesta() {
138                 if (this.estatus)
139                 {
140                         return mensajeExitoso();
141                 } else {
142                         return mensajeError();
143                 }// end if
144         }
145
146         public SOAPMessage mensajeExitoso() {
147         SOAPMessage message = null;
148
149         try {
150             // create price list message
151             message = fac.createMessage();
152
153             // Access the SOAPBody object
154             SOAPPart part = message.getSOAPPart();
155             SOAPEnvelope envelope = part.getEnvelope();
156             SOAPBody body = envelope.getBody();
157                         SOAPHeader header = envelope.getHeader();
158                         header.detachNode();
159
160             Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
161                 "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
162             SOAPBodyElement list = body.addBodyElement(bodyName);
163
164             Name exitosoN = envelope.createName("Respuesta");
165             SOAPElement exitoso = list.addChildElement(exitosoN);
166                         exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
167             message.saveChanges();
168
169         } catch(Exception e) {
170             e.printStackTrace();
171         }
172         return message;
173       }// end mensajeExitoso
174
175
176         public SOAPMessage mensajeError() {
177         SOAPMessage message = null;
178
179         try {
180             // create price list message
181             message = fac.createMessage();
182
183             // Access the SOAPBody object
184             SOAPPart part = message.getSOAPPart();
185             SOAPEnvelope envelope = part.getEnvelope();
186             SOAPBody body = envelope.getBody();
187                         SOAPHeader header = envelope.getHeader();
188                         header.detachNode();
189
190                         SOAPFactory soapFactory = SOAPFactory.newInstance();
```

```
191                             SOAPFault fault = body.addFault();
192                             Name faultName =
193                                     soapFactory.createName("Server",
194                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
195                             fault.setFaultCode(faultName);
196                             fault.setFaultActor("http://www.unam.mx/AltaUnidad"); //el mismo nombre de la clase
197                             fault.setFaultString(this.codigoError);
198                 message.saveChanges();
199
200             } catch(Exception e) {
201                 e.printStackTrace();
202             }
203             return message;
204         }// end servicioError
205
206
207             public String getAgenciaId() {
208                     return this.agenciaid;
209             }
210
211             public String getPassword() {
212                     return this.password;
213             }
214
215             public String getSerieVehiculo() {
216                     return this.serieVehiculo;
217             }
218
219             public boolean getEstatus() {
220                     return this.estatus;
221             }
222
223             public String getCodigoError(){
224                     return this.codigoError;
225             }
226
227             public void setCodigoError(String codigoError) {
228                     this.codigoError=codigoError;
229             }
230
231     }
```

AltaVendedor.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class AltaVendedor
16      {
17              private String agenciaid="";
18              private String password="";
19              private String rfcVendedor="";
20              private String nombre="";
21              private boolean estatus = false;
22              private String codigoError = "5010-No hay datos para procesar";
23          static MessageFactory fac = null;
24
25          static {
26              try {
27                  fac = MessageFactory.newInstance();
28              } catch (Exception ex) {
29                  ex.printStackTrace();
30              }
31          }
32
33              public AltaVendedor(MimeHeaders headers, InputStream is)   throws IOException {
34                      try
```

```
35                      {
36                              SOAPMessage msg = fac.createMessage(headers, is);
37                  SOAPPart part = msg.getSOAPPart();
38
39                              // Código Agregado para la versión 2.0 del sistema
40                              // para validar datos desde la llegada del xml
41                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
42                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/AltaVendedor.xsd");
43                              Verifier verifier = schema.newVerifier();
44                              verifier.verify(part);
45                              // fin de código para version 2.0
46
47                  SOAPEnvelope envelope = part.getEnvelope();
48                  SOAPBody body = envelope.getBody();
49                              java.util.Iterator iterator = body.getChildElements();
50                              SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
51                              iterator = addUnit.getChildElements();
52                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
53                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
54                              SOAPBodyElement rfcVendedor = (SOAPBodyElement)iterator.next();
55                              SOAPBodyElement nombre = (SOAPBodyElement)iterator.next();
56                              this.agenciaid=agenciaid.getValue();
57                              this.password=password.getValue();
58                              this.rfcVendedor =rfcVendedor.getValue();
59                              this.nombre =nombre.getValue();
60                      }
61                      catch (SOAPException e)
62                      {
63                              this.estatus=false;
64                              this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
65                      }
66                      catch (VerifierConfigurationException e) // catch requerido para version 2.0
67                      {
68                              this.estatus=false;
69                              this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
70                      }
71                      catch (SAXException e) // catch requerido para version 2.0
72                      {
73                              this.estatus=false;
74                              this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
75                      }
76              }
77
78              public AltaVendedor(String agenciaid, String password, String rfcVendedor, String nombre) {
79                      this.agenciaid=agenciaid;
80                      this.password=password;
81                      this.rfcVendedor=rfcVendedor;
82                      this.nombre=nombre;
83              }
84
85              public boolean ejecuta(ConnectionPool connectionPool) {
86                      String strSQL;
87                      // if requerido para version 2.0
88                      if (this.codigoError.equals( "5010-No hay datos para procesar"))
89                      {
90
91                      try {
92
93                              Connection miconexion = connectionPool.getConnection();
94                              String comilla ="'";
95                              ResultSet rs ,rs2;
96                              String serieVehiculo;
97                              PreparedStatement pstmInsert;
98                              Statement stm = miconexion.createStatement();
99                              strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
100                                      "WHERE agenciaid = " + comilla + agenciaid + comilla +
101                                          "AND password = password("+comilla + password + comilla +")";
102                             rs = stm.executeQuery(strSQL);
103                             rs.next();
104                             if (rs.getInt("conteo")!=1)
105                             {
106                                     this.estatus=false;
107                                 this.codigoError="5000 - agenciaid:" + agenciaid +
108                                                 " con password:" + password + " no son validos. Contacte al administrador ";
109                             } else {
110                                     this.estatus=true;
111                                     strSQL = "SELECT count(*) as conteo FROM tblVendedores " +
112                              " WHERE rfcVendedor = " + comilla + this.rfcVendedor + comilla +
```

```
113                          " AND agenciaid = " + comilla + this.agenciaid + comilla ;
114                                  rs = stm.executeQuery(strSQL);
115                                  rs.next();
116                                  if (rs.getInt("conteo")==0) {
117                                          strSQL = "INSERT INTO tblVendedores(rfcVendedor,agenciaid,nombre) VALUES "+
118                                                          "(" + comilla + this.rfcVendedor + comilla +
119                                                          "," + comilla + this.agenciaid + comilla +
120                                                          "," + comilla + this.nombre + comilla +")";
121                                          pstmInsert = miconexion.prepareStatement(strSQL);
122                                          pstmInsert.execute();
123                                          pstmInsert.close();
124                                  }// end if
125                          }
126                          rs.close();
127                          miconexion.close();
128                  } catch(SQLException sqle) {
129                    switch (sqle.getErrorCode())
130                    {
131                    case 0:
132                      this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
133                          break;
134                    case 1049:
135                      this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
136                          break;
137                    case 1216: //para fks multiples se tendra que revisar que datos no son validos
138                        this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                    administrador ";
139                          break;
140                    default:
141                            this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                    Contacte al administrador ";
142                    }
143                    //connectionPool = null;
144                    this.estatus=false;
145                  }// end catch
146                  }// end if
147                  return (this.estatus);
148          }
149
150
151          public SOAPMessage respuesta() {
152                  if (this.estatus)
153                  {
154                          return mensajeExitoso();
155                  } else {
156                          return mensajeError();
157                  }// end if
158          }
159
160          public SOAPMessage mensajeExitoso() {
161          SOAPMessage message = null;
162
163          try {
164              // create price list message
165              message = fac.createMessage();
166
167              // Access the SOAPBody object
168              SOAPPart part = message.getSOAPPart();
169              SOAPEnvelope envelope = part.getEnvelope();
170              SOAPBody body = envelope.getBody();
171                          SOAPHeader header = envelope.getHeader();
172                          header.detachNode();
173
174              Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
175                  "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
176              SOAPBodyElement list = body.addBodyElement(bodyName);
177
178               Name exitosoN = envelope.createName("Respuesta");
179              SOAPElement exitoso = list.addChildElement(exitosoN);
180                          exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
181              message.saveChanges();
182
183          } catch(Exception e) {
184              e.printStackTrace();
185          }
186          return message;
187        }// end mensajeExitoso
188
189              public SOAPMessage mensajeError() {
```

```
190       SOAPMessage message = null;
191
192           try {
193               // create price list message
194               message = fac.createMessage();
195
196
197               // Access the SOAPBody object
198               SOAPPart part = message.getSOAPPart();
199               SOAPEnvelope envelope = part.getEnvelope();
200               SOAPBody body = envelope.getBody();
201                           SOAPHeader header = envelope.getHeader();
202                           header.detachNode();
203
204                           SOAPFactory soapFactory = SOAPFactory.newInstance();
205                           SOAPFault fault = body.addFault();
206                           Name faultName =
207                                   soapFactory.createName("Server",
208                                   "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
209                           fault.setFaultCode(faultName);
210                            fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
211                           fault.setFaultString(this.codigoError);
212               message.saveChanges();
213
214           } catch(Exception e) {
215               e.printStackTrace();
216           }
217           return message;
218       }// end servicioError
219
220
221           public String getAgenciaId() {
222                   return this.agenciaid;
223           }
224
225           public String getPassword() {
226                   return this.password;
227           }
228
229           public boolean getEstatus() {
230                   return this.estatus;
231           }
232
233           public String getCodigoError(){
234                   return this.codigoError;
235           }
236
237           public void setCodigoError(String codigoError) {
238                   this.codigoError=codigoError;
239           }
240
241   }
```

BajaAsesor.java

```
1     // Bibliotecas para validar el mensaje XML SOAP
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
8     import javax.xml.soap.*;
9     import java.util.*;
10
11    // Bibliotecas para manejar la conexión de sql
12    import java.sql.*;
13    import coreservlets.*;
14
15    public class BajaAsesor
16    {
17            private String agenciaid="";
18            private String password="";
19            private String rfcAsesor ="";
20            private boolean estatus = false;
21            private String codigoError = "5010-No hay datos para procesar";
```

```
22      static MessageFactory fac = null;
23
24      static {
25          try {
26              fac = MessageFactory.newInstance();
27          } catch (Exception ex) {
28              ex.printStackTrace();
29          }
30      }
31
32          public BajaAsesor(MimeHeaders headers, InputStream is)     throws IOException {
33                  try
34                  {
35                          SOAPMessage msg = fac.createMessage(headers, is);
36              SOAPPart part = msg.getSOAPPart();
37
38                          // Código Agregado para la versión 2.0 del sistema
39                          // para validar datos desde la llegada del xml
40                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                  Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/BajaAsesor.xsd");
42                          Verifier verifier = schema.newVerifier();
43                          verifier.verify(part);
44                          // fin de código para version 2.0
45
46                          SOAPEnvelope envelope = part.getEnvelope();
47              SOAPBody body = envelope.getBody();
48                          java.util.Iterator iterator = body.getChildElements();
49                          SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                          iterator = addUnit.getChildElements();
51                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement rfcVendedor = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                          this.rfcAsesor = rfcVendedor.getValue();
57                  }
58                  catch (SOAPException e)
59                  {
60                          this.estatus=false;
61                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
62                  }
63                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                  {
65                          this.estatus=false;
66                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                  }
68                  catch (SAXException e) // catch requerido para version 2.0
69                  {
70                          this.estatus=false;
71                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                  }
73
74          }
75
76          public BajaAsesor(String agenciaid, String password, String rfcAsesor) {
77                  this.agenciaid=agenciaid;
78                  this.password=password;
79                  this.rfcAsesor=rfcAsesor;
80          }
81
82          public boolean ejecuta(ConnectionPool connectionPool) {
83                  String strSQL;
84                  // if requerido para version 2.0
85                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
86                  {
87                  try {
88
89                          Connection miconexion = connectionPool.getConnection();
90                          String comilla ="'";
91                          ResultSet rs ,rs2;
92                          String serieVehiculo;
93                          PreparedStatement pstm;
94                          Statement stm = miconexion.createStatement();
95                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
96                                   "WHERE agenciaid = " + comilla + agenciaid + comilla +
97                                       "AND password = password("+comilla + password + comilla +")";
98                          rs = stm.executeQuery(strSQL);
99                          rs.next();
```

```
100                         if (rs.getInt("conteo")!=1)
101                         {
102                                 this.estatus=false;
103                             this.codigoError="5000 - agenciaid:" + agenciaid +
104                                         " con password:" + password + " no son validos. Contacte al administrador ";
105                         } else {
106                                 strSQL = "SELECT count(*) as conteo FROM tblOrdenes " +
107                                                 " WHERE rfcAsesor = " + comilla + this.rfcAsesor + comilla +
108                                                 " AND agenciaid = "+comilla + this.agenciaid + comilla ;
109                                 rs = stm.executeQuery(strSQL);
110                                 rs.next();
111                                 if (rs.getInt("conteo")>0) {
112                                         this.estatus=false;
113                                     this.codigoError="5200 - El asesor " + this.rfcAsesor +
114                                               " de la agencia " + this.agenciaid + " no se puede eliminar. Esta siendo
                                                                                                 utilizado en Ordenes de Reparación ";
115                                 } else {
116                                         this.estatus=true;
117                                         strSQL = "DELETE FROM tblAsesores "+
118                                                         " WHERE rfcAsesor = " + comilla + this.rfcAsesor + comilla +
119                                                         " AND agenciaid=" + comilla + this.agenciaid + comilla ;
120                                         pstm = miconexion.prepareStatement(strSQL);
121                                         pstm.execute();
122                                         pstm.close();
123                                 } // end if
124                         }// end if
125                         rs.close();
126                         miconexion.close();
127                 } catch(SQLException sqle) {
128                   switch (sqle.getErrorCode())
129                   {
130                   case 0:
131                     this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
132                         break;
133                   case 1049:
134                     this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
135                         break;
136                   case 1216: //para fks multiples se tendra que revisar que datos no son validos
137                       this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                 administrador ";
138                         break;
139                   default:
140                           this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                 Contacte al administrador ";
141                   } // end switch
142                   //connectionPool = null;
143                   this.estatus=false;
144                 }// end catch
145                 } //end if
146                 return (this.estatus);
147           }
148
149
150           public SOAPMessage respuesta() {
151                   if (this.estatus)
152                   {
153                           return mensajeExitoso();
154                   } else {
155                           return mensajeError();
156                   }// end if
157           }
158
159           public SOAPMessage mensajeExitoso() {
160           SOAPMessage message = null;
161
162           try {
163               // create price list message
164               message = fac.createMessage();
165
166               // Access the SOAPBody object
167               SOAPPart part = message.getSOAPPart();
168               SOAPEnvelope envelope = part.getEnvelope();
169               SOAPBody body = envelope.getBody();
170                           SOAPHeader header = envelope.getHeader();
171                           header.detachNode();
172
173               Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
174                   "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
175               SOAPBodyElement list = body.addBodyElement(bodyName);
```

```
176
177                 Name exitosoN = envelope.createName("Respuesta");
178                 SOAPElement exitoso = list.addChildElement(exitosoN);
179                             exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
180                 message.saveChanges();
181
182             } catch(Exception e) {
183                 e.printStackTrace();
184             }
185             return message;
186         }// end mensajeExitoso
187
188             public SOAPMessage mensajeError() {
189             SOAPMessage message = null;
190
191             try {
192                 // create price list message
193                 message = fac.createMessage();
194
195
196                 // Access the SOAPBody object
197                 SOAPPart part = message.getSOAPPart();
198                 SOAPEnvelope envelope = part.getEnvelope();
199                 SOAPBody body = envelope.getBody();
200                             SOAPHeader header = envelope.getHeader();
201                             header.detachNode();
202
203                             SOAPFactory soapFactory = SOAPFactory.newInstance();
204                             SOAPFault fault = body.addFault();
205                             Name faultName =
206                                     soapFactory.createName("Server",
207                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
208                             fault.setFaultCode(faultName);
209                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
210                             fault.setFaultString(this.codigoError);
211                 message.saveChanges();
212
213             } catch(Exception e) {
214                 e.printStackTrace();
215             }
216             return message;
217         }// end servicioError
218
219
220             public String getAgenciaId() {
221                     return this.agenciaid;
222             }
223
224             public String getPassword() {
225                     return this.password;
226             }
227
228             public boolean getEstatus() {
229                     return this.estatus;
230             }
231
232             public String getCodigoError(){
233                     return this.codigoError;
234             }
235
236             public void setCodigoError(String codigoError) {
237                     this.codigoError=codigoError;
238             }
239
240     }
```

BajaCliente.java

```
1     // Bibliotecas para validar el mensaje XML SOAP
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
```

```
8   import javax.xml.soap.*;
9   import java.util.*;
10
11  // Bibliotecas para manejar la conexión de sql
12  import java.sql.*;
13  import coreservlets.*;
14
15  public class BajaCliente
16  {
17          private String agenciaid="";
18          private String password="";
19          private String rfc_o_curp_cliente ="";
20          private boolean estatus = false;
21          private String codigoError = "5010-No hay datos para procesar";
22      static MessageFactory fac = null;
23
24      static {
25          try {
26              fac = MessageFactory.newInstance();
27          } catch (Exception ex) {
28              ex.printStackTrace();
29          }
30      }
31
32          public BajaCliente(MimeHeaders headers, InputStream is)    throws IOException {
33                  try
34                  {
35                          SOAPMessage msg = fac.createMessage(headers, is);
36                          SOAPPart part = msg.getSOAPPart();
37
38                          // Código Agregado para la versión 2.0 del sistema
39                          // para validar datos desde la llegada del xml
40                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                  Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/BajaCliente.xsd");
42                          Verifier verifier = schema.newVerifier();
43                          verifier.verify(part);
44                          // fin de código para version 2.0
45
46                          SOAPEnvelope envelope = part.getEnvelope();
47                          SOAPBody body = envelope.getBody();
48                          java.util.Iterator iterator = body.getChildElements();
49                          SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                          iterator = addUnit.getChildElements();
51                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement rfc_o_curp_cliente = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                          this.rfc_o_curp_cliente = rfc_o_curp_cliente.getValue();
57                  }
58                  catch (SOAPException e)
59                  {
60                          this.estatus=false;
61                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
62                  }
63                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                  {
65                          this.estatus=false;
66                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                  }
68                  catch (SAXException e) // catch requerido para version 2.0
69                  {
70                          this.estatus=false;
71                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                  }
73
74          }
75
76          public BajaCliente(String agenciaid, String password, String rfc_o_curp_cliente) {
77                  this.agenciaid=agenciaid;
78                  this.password=password;
79                  this.rfc_o_curp_cliente = rfc_o_curp_cliente;
80          }
81
82          public boolean ejecuta(ConnectionPool connectionPool) {
83                  String strSQL;
84                  // if requerido para version 2.0
85                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
```

```
86                      {
87                      try {
88
89                              Connection miconexion = connectionPool.getConnection();
90                              String comilla ="'";
91                              ResultSet rs ,rs2;
92                              String serieVehiculo;
93                              PreparedStatement pstm;
94                              Statement stm = miconexion.createStatement();
95                              strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
96                                       "WHERE agenciaid = " + comilla + agenciaid + comilla +
97                                            "AND password = password("+comilla + password + comilla +")";
98                              rs = stm.executeQuery(strSQL);
99                              rs.next();
100                             if (rs.getInt("conteo")!=1) // if #1
101                             {
102                                     this.estatus=false;
103                                 this.codigoError="5000 - agenciaid:" + agenciaid +
104                                             " con password:" + password + " no son validos. Contacte al administrador ";
105                             } else {
106                                     strSQL = "SELECT count(*) as conteo FROM tblVentas " +
107                                 "WHERE agenciaid = " + comilla + this.agenciaid + comilla +
108                                     "AND rfc_o_curp_cliente = " + comilla + this.rfc_o_curp_cliente + comilla ;
109                                     rs = stm.executeQuery(strSQL);
110                                     rs.next();
111                                     if (rs.getInt("conteo")>0) { // if #2
112                                             this.estatus=false;
113                                         this.codigoError="5100 - cliente " + this.rfc_o_curp_cliente +
114                                                     " en agencia " + this.agenciaid + " esta siendo utilizado en ventas de
                                                                                                   vehiculos y no puede eliminarse ";
115                                     } else {
116                                             strSQL = "SELECT count(*) as conteo FROM tblOrdenes " +
117                                             "WHERE agenciaid = " + comilla + this.agenciaid + comilla +
118                                             "AND rfc_o_curp_cliente = " + comilla + this.rfc_o_curp_cliente + comilla ;
119                                             rs = stm.executeQuery(strSQL);
120                                             rs.next();
121                                             if (rs.getInt("conteo")>0) {// if #3
122                                                     this.estatus=false;
123                                                     this.codigoError="5110 - cliente " + this.rfc_o_curp_cliente +
124                                                       " en agencia " + this.agenciaid + " esta siendo utilizado en ordenes
                                                                                                 de reparacion y no puede eliminarse ";
125                                             } else {
126                                                     this.estatus=true;
127                                                     strSQL = "DELETE FROM tblTelefonos "+
128                                                                                     "WHERE rfc_o_curp_cliente = " + comilla +
                                                                                           this.rfc_o_curp_cliente + comilla +
129                                                                                "AND agenciaid=" + comilla + this.agenciaid +
                                                                                                                   comilla ;
130                                                     pstm = miconexion.prepareStatement(strSQL);
131                                                     pstm.execute();
132                                                     strSQL = "DELETE FROM tblClientes "+
133                                                                                     "WHERE rfc_o_curp_cliente = " + comilla +
                                                                                           this.rfc_o_curp_cliente + comilla +
134                                                                                "AND agenciaid=" + comilla + this.agenciaid +
                                                                                                                   comilla ;
135                                                     pstm = miconexion.prepareStatement(strSQL);
136                                                     pstm.execute();
137                                                     pstm.close();
138                                             }// end if #3
139                                     }// end if #2
140                             }//end if #1
141                             rs.close();
142                             miconexion.close();
143                     } catch(SQLException sqle) {
144                       switch (sqle.getErrorCode())
145                       {
146                       case 0:
147                         this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
148                              break;
149                       case 1049:
150                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
151                              break;
152                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
153                            this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                 administrador ";
154                              break;
155                       default:
156                                this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                   Contacte al administrador ";
```

```
157                     }  // end switch
158                     //connectionPool = null;
159                     this.estatus=false;
160                   } // end catch
161                   } // end if
162                   return (this.estatus);
163           }
164
165
166           public SOAPMessage respuesta() {
167                   if (this.estatus)
168                   {
169                           return mensajeExitoso();
170                   } else {
171                           return mensajeError();
172                   }// end if
173           }
174
175           public SOAPMessage mensajeExitoso() {
176       SOAPMessage message = null;
177
178           try {
179               // create price list message
180               message = fac.createMessage();
181
182               // Access the SOAPBody object
183               SOAPPart part = message.getSOAPPart();
184               SOAPEnvelope envelope = part.getEnvelope();
185               SOAPBody body = envelope.getBody();
186                           SOAPHeader header = envelope.getHeader();
187                           header.detachNode();
188
189               Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
190                   "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
191               SOAPBodyElement list = body.addBodyElement(bodyName);
192
193               Name exitosoN = envelope.createName("Respuesta");
194               SOAPElement exitoso = list.addChildElement(exitosoN);
195                           exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
196               message.saveChanges();
197
198
199           } catch(Exception e) {
200               e.printStackTrace();
201           }
202           return message;
203       }// end mensajeExitoso
204
205           public SOAPMessage mensajeError() {
206       SOAPMessage message = null;
207
208           try {
209               // create price list message
210               message = fac.createMessage();
211
212
213               // Access the SOAPBody object
214               SOAPPart part = message.getSOAPPart();
215               SOAPEnvelope envelope = part.getEnvelope();
216               SOAPBody body = envelope.getBody();
217                           SOAPHeader header = envelope.getHeader();
218                           header.detachNode();
219
220                           SOAPFactory soapFactory = SOAPFactory.newInstance();
221                           SOAPFault fault = body.addFault();
222                           Name faultName =
223                                   soapFactory.createName("Server",
224                                   "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
225                           fault.setFaultCode(faultName);
226                             fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
227                           fault.setFaultString(this.codigoError);
228               message.saveChanges();
229
230           } catch(Exception e) {
231               e.printStackTrace();
232           }
233           return message;
234       }// end servicioError
```

```
235
236
237             public String getAgenciaId() {
238                     return this.agenciaid;
239             }
240
241             public String getPassword() {
242                     return this.password;
243             }
244
245             public boolean getEstatus() {
246                     return this.estatus;
247             }
248
249             public String getCodigoError(){
250                     return this.codigoError;
251             }
252
253             public void setCodigoError(String codigoError) {
254                     this.codigoError=codigoError;
255             }
256
257     }
```

BajaUnidad.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class BajaUnidad
16      {
17              private String agenciaid="";
18              private String password="";
19              private String serieVehiculo="";
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22          static MessageFactory fac = null;
23
24          static {
25              try {
26                  fac = MessageFactory.newInstance();
27              } catch (Exception ex) {
28                  ex.printStackTrace();
29              }
30          }
31
32              public BajaUnidad(MimeHeaders headers, InputStream is)     throws IOException {
33                      try
34                      {
35                              SOAPMessage msg = fac.createMessage(headers, is);
36                  SOAPPart part = msg.getSOAPPart();
37
38                              // Código Agregado para la versión 2.0 del sistema
39                              // para validar datos desde la llegada del xml
40                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/BajaUnidad.xsd");
42                              Verifier verifier = schema.newVerifier();
43                              verifier.verify(part);
44                              // fin de código para version 2.0
45
46                              SOAPEnvelope envelope = part.getEnvelope();
47                  SOAPBody body = envelope.getBody();
48                              java.util.Iterator iterator = body.getChildElements();
49                              SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
```

```
50                          iterator = addUnit.getChildElements();
51                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                          this.serieVehiculo = serieVehiculo.getValue();
57                  }
58                  catch (SOAPException e)
59                  {
60                          this.estatus=false;
61                          this.codigoError="6000-Error al leer el xml de entrada en AltaUnidad";
62                  }
63                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                  {
65                          this.estatus=false;
66                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                  }
68                  catch (SAXException e) // catch requerido para version 2.0
69                  {
70                          this.estatus=false;
71                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                  }
73
74          }
75
76          public BajaUnidad(String agenciaid, String password, String serieVehiculo) {
77                  this.agenciaid=agenciaid;
78                  this.password=password;
79                  this.serieVehiculo=serieVehiculo;
80          }
81
82          public boolean ejecuta(ConnectionPool connectionPool) {
83                  String strSQL;
84                  // if requerido para version 2.0
85                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
86                  {
87
88                  try {
89
90                          Connection miconexion = connectionPool.getConnection();
91                          String comilla ="'";
92                          Statement stm = miconexion.createStatement();
93                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
94                                   "WHERE agenciaid = " + comilla + agenciaid + comilla +
95                                        "AND password = password("+comilla + password + comilla +")";
96                          ResultSet rs = stm.executeQuery(strSQL);
97                          rs.next();
98                          if (rs.getInt("conteo")!=1)
99                          {
100                                 this.estatus=false;
101                             this.codigoError="5000 - agenciaid:" + agenciaid +
102                                         " con password:" + password + " no son validos. Contacte al administrador ";
103                         } else {
104                                 strSQL = "SELECT count(*) as conteo FROM tblVentas " +
105                                          "WHERE serieVehiculo = " + comilla + this.serieVehiculo + comilla ;
106                                 rs = stm.executeQuery(strSQL);
107                                 rs.next();
108                                 if (rs.getInt("conteo")==1){
109                                         this.estatus=false;
110                                                 this.codigoError="5210 - No se puede eliminar el vehiculo " +
                                                                                                 this.serieVehiculo +
111                                                         " del inventario. Ya se ha reportado la venta de este vehiculo";
112                                 } else {
113                                         this.estatus=true;
114                                                strSQL = "DELETE FROM tblInventario WHERE serievehiculo = " + comilla +
                                                                                                 this.serieVehiculo + comilla +
115                                                          " AND agenciaid=" + comilla + this.agenciaid + comilla ;
116                                         PreparedStatement pstmInsert = miconexion.prepareStatement(strSQL);
117                                         pstmInsert.execute();
118                                         pstmInsert.close();
119                                 }
120                         }
121                         rs.close();
122                         miconexion.close();
123                 } catch(SQLException sqle) {
124                   switch (sqle.getErrorCode())
125                   {
126                   case 0:
```

```
127                       this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
128                            break;
129                       case 1049:
130                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
131                            break;
132                       case 1062:
133                         this.codigoError="1062 - Unidad " + this.serieVehiculo + " ya se encuentra en Inventario ";
134                            break;
135                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
136                           this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                    administrador ";
137                            break;
138                       default:
139                               this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                    Contacte al administrador ";
140                       } //end switch
141                       //connectionPool = null;
142                       this.estatus=false;
143                     }//end catch
144                     }//end if
145                     return (this.estatus);
146             }
147
148
149             public SOAPMessage respuesta() {
150                     if (this.estatus)
151                     {
152                             return mensajeExitoso();
153                     } else {
154                             return mensajeError();
155                     }// end if
156             }
157
158             public SOAPMessage mensajeExitoso() {
159           SOAPMessage message = null;
160
161             try {
162                 // create price list message
163                 message = fac.createMessage();
164
165                 // Access the SOAPBody object
166                 SOAPPart part = message.getSOAPPart();
167                 SOAPEnvelope envelope = part.getEnvelope();
168                 SOAPBody body = envelope.getBody();
169                             SOAPHeader header = envelope.getHeader();
170                             header.detachNode();
171
172                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
173                     "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
174                 SOAPBodyElement list = body.addBodyElement(bodyName);
175
176                 Name exitosoN = envelope.createName("Respuesta");
177                 SOAPElement exitoso = list.addChildElement(exitosoN);
178                             exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
179                 message.saveChanges();
180
181             } catch(Exception e) {
182                 e.printStackTrace();
183             }
184             return message;
185           }// end mensajeExitoso
186
187
188             public SOAPMessage mensajeError() {
189             SOAPMessage message = null;
190
191             try {
192                 // create price list message
193                 message = fac.createMessage();
194
195
196                 // Access the SOAPBody object
197                 SOAPPart part = message.getSOAPPart();
198                 SOAPEnvelope envelope = part.getEnvelope();
199                 SOAPBody body = envelope.getBody();
200                             SOAPHeader header = envelope.getHeader();
201                             header.detachNode();
202
203                             SOAPFactory soapFactory = SOAPFactory.newInstance();
```

```
204                             SOAPFault fault = body.addFault();
205                             Name faultName =
206                                     soapFactory.createName("Server",
207                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
208                             fault.setFaultCode(faultName);
209                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
210                             fault.setFaultString(this.codigoError);
211                 message.saveChanges();
212
213             } catch(Exception e) {
214                 e.printStackTrace();
215             }
216             return message;
217         }// end servicioError
218
219
220             public String getAgenciaId() {
221                     return this.agenciaid;
222             }
223
224             public String getPassword() {
225                     return this.password;
226             }
227
228             public String getSerieVehiculo() {
229                     return this.serieVehiculo;
230             }
231
232             public boolean getEstatus() {
233                     return this.estatus;
234             }
235
236             public String getCodigoError(){
237                     return this.codigoError;
238             }
239
240             public void setCodigoError(String codigoError) {
241                     this.codigoError=codigoError;
242             }
243
244     }
```

BajaVendedor.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class BajaVendedor
16      {
17              private String agenciaid="";
18              private String password="";
19              private String rfcVendedor ="";
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22          static MessageFactory fac = null;
23
24          static {
25              try {
26                  fac = MessageFactory.newInstance();
27              } catch (Exception ex) {
28                  ex.printStackTrace();
29              }
30          }
31
32              public BajaVendedor(MimeHeaders headers, InputStream is)   throws IOException {
33                      try
```

```
34                      {
35                              SOAPMessage msg = fac.createMessage(headers, is);
36                  SOAPPart part = msg.getSOAPPart();
37
38                              // Código Agregado para la versión 2.0 del sistema
39                              // para validar datos desde la llegada del xml
40                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/BajaVendedor.xsd");
42                              Verifier verifier = schema.newVerifier();
43                              verifier.verify(part);
44                              // fin de código para version 2.0
45
46                              SOAPEnvelope envelope = part.getEnvelope();
47                  SOAPBody body = envelope.getBody();
48                              java.util.Iterator iterator = body.getChildElements();
49                              SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                              iterator = addUnit.getChildElements();
51                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                              SOAPBodyElement rfcVendedor = (SOAPBodyElement)iterator.next();
54                              this.agenciaid=agenciaid.getValue();
55                              this.password=password.getValue();
56                              this.rfcVendedor = rfcVendedor.getValue();
57                      }
58                      catch (SOAPException e)
59                      {
60                              this.estatus=false;
61                              this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
62                      }
63                      catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                      {
65                              this.estatus=false;
66                              this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                      }
68                      catch (SAXException e) // catch requerido para version 2.0
69                      {
70                              this.estatus=false;
71                              this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                      }
73
74              }
75
76              public BajaVendedor(String agenciaid, String password, String rfcVendedor) {
77                      this.agenciaid=agenciaid;
78                      this.password=password;
79                      this.rfcVendedor=rfcVendedor;
80              }
81
82              public boolean ejecuta(ConnectionPool connectionPool) {
83                      String strSQL;
84                      // if requerido para version 2.0
85                      if (this.codigoError.equals( "5010-No hay datos para procesar"))
86                      {
87                      try {
88
89                              Connection miconexion = connectionPool.getConnection();
90                              String comilla ="'";
91                              ResultSet rs ,rs2;
92                              String serieVehiculo;
93                              PreparedStatement pstm;
94                              Statement stm = miconexion.createStatement();
95                              strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
96                                       "WHERE agenciaid = " + comilla + agenciaid + comilla +
97                                            "AND password = password("+comilla + password + comilla +")";
98                              rs = stm.executeQuery(strSQL);
99                              rs.next();
100                             if (rs.getInt("conteo")!=1)
101                             {
102                                     this.estatus=false;
103                                 this.codigoError="5000 - agenciaid:" + agenciaid +
104                                                  " con password:" + password + " no son validos. Contacte al administrador ";
105                             } else {
106                                     strSQL = "SELECT count(*) as conteo FROM tblVentas " +
107                                                      " WHERE rfcVendedor = " + comilla + this.rfcVendedor + comilla +
108                                                      " AND agenciaid = "+comilla + this.agenciaid + comilla ;
109                                     rs = stm.executeQuery(strSQL);
110                                     rs.next();
111                                     if (rs.getInt("conteo")>0) {
```

```
112                                     this.estatus=false;
113                                   this.codigoError="5200 - El vendedor " + this.rfcVendedor +
114                                           " de la agencia " + this.agenciaid + " no se puede eliminar. Esta siendo
                                                                                    utilizado en informacion de Ventas ";
115                                 } else {
116                                     this.estatus=true;
117                                     strSQL = "DELETE FROM tblVendedores "+
118                                                                 " WHERE rfcVendedor = " + comilla + this.rfcVendedor +
                                                                                                        comilla +
119                                                             " AND agenciaid=" + comilla + this.agenciaid + comilla ;
120                                     pstm = miconexion.prepareStatement(strSQL);
121                                     pstm.execute();
122                                     pstm.close();
123                                 } // end if
124                             }// end if
125                             rs.close();
126                             miconexion.close();
127                     } catch(SQLException sqle) {
128                       switch (sqle.getErrorCode())
129                       {
130                       case 0:
131                         this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
132                             break;
133                       case 1049:
134                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
135                             break;
136                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
137                           this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                        administrador ";
138                             break;
139                       default:
140                                this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                    Contacte al administrador ";
141                       }// end switch
142                       //connectionPool = null;
143                       this.estatus=false;
144                     }// end catch
145                     }// end if
146                     return (this.estatus);
147             }
148
149
150             public SOAPMessage respuesta() {
151                     if (this.estatus)
152                     {
153                             return mensajeExitoso();
154                     } else {
155                             return mensajeError();
156                     }// end if
157             }
158
159             public SOAPMessage mensajeExitoso() {
160         SOAPMessage message = null;
161
162             try {
163                 // create price list message
164                 message = fac.createMessage();
165
166                 // Access the SOAPBody object
167                 SOAPPart part = message.getSOAPPart();
168                 SOAPEnvelope envelope = part.getEnvelope();
169                 SOAPBody body = envelope.getBody();
170                             SOAPHeader header = envelope.getHeader();
171                             header.detachNode();
172
173                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
174                     "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
175                 SOAPBodyElement list = body.addBodyElement(bodyName);
176
177                 Name exitosoN = envelope.createName("Respuesta");
178                 SOAPElement exitoso = list.addChildElement(exitosoN);
179                             exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
180                 message.saveChanges();
181
182             } catch(Exception e) {
183                 e.printStackTrace();
184             }
185             return message;
186         }// end mensajeExitoso
```

```
187
188             public SOAPMessage mensajeError() {
189         SOAPMessage message = null;
190
191             try {
192                 // create price list message
193                 message = fac.createMessage();
194
195
196                 // Access the SOAPBody object
197                 SOAPPart part = message.getSOAPPart();
198                 SOAPEnvelope envelope = part.getEnvelope();
199                 SOAPBody body = envelope.getBody();
200                             SOAPHeader header = envelope.getHeader();
201                             header.detachNode();
202
203                             SOAPFactory soapFactory = SOAPFactory.newInstance();
204                             SOAPFault fault = body.addFault();
205                             Name faultName =
206                                     soapFactory.createName("Server",
207                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
208                             fault.setFaultCode(faultName);
209                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
210                             fault.setFaultString(this.codigoError);
211                 message.saveChanges();
212
213             } catch(Exception e) {
214                 e.printStackTrace();
215             }
216             return message;
217         }// end servicioError
218
219
220             public String getAgenciaId() {
221                     return this.agenciaid;
222             }
223
224             public String getPassword() {
225                     return this.password;
226             }
227
228             public boolean getEstatus() {
229                     return this.estatus;
230             }
231
232             public String getCodigoError(){
233                     return this.codigoError;
234             }
235
236             public void setCodigoError(String codigoError) {
237                     this.codigoError=codigoError;
238             }
239
240     }
```

CancelaOrden.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class CancelaOrden
16      {
17              private String agenciaid="";
18              private String password="";
```

```
19              private String ordenid="";
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22          static MessageFactory fac = null;
23
24          static {
25              try {
26                  fac = MessageFactory.newInstance();
27              } catch (Exception ex) {
28                  ex.printStackTrace();
29              }
30          }
31
32              public CancelaOrden(MimeHeaders headers, InputStream is)   throws IOException {
33                      try
34                      {
35                              SOAPMessage msg = fac.createMessage(headers, is);
36                              SOAPPart part = msg.getSOAPPart();
37
38                              // Código Agregado para la versión 2.0 del sistema
39                              // para validar datos desde la llegada del xml
40                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/CancelaOrden.xsd");
42                              Verifier verifier = schema.newVerifier();
43                              verifier.verify(part);
44
45                              // fin de código para version 2.0
46                              SOAPEnvelope envelope = part.getEnvelope();
47                              SOAPBody body = envelope.getBody();
48                              java.util.Iterator iterator = body.getChildElements();
49                              SOAPBodyElement cierraOrden = (SOAPBodyElement)iterator.next();
50                              iterator = cierraOrden.getChildElements();
51                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                              SOAPBodyElement ordenid = (SOAPBodyElement)iterator.next();
54                              this.agenciaid=agenciaid.getValue();
55                              this.password=password.getValue();
56                              this.ordenid = ordenid.getValue();
57                      }
58                      catch (SOAPException e)
59                      {
60                              this.estatus=false;
61                              this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
62                      }
63                      catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                      {
65                              this.estatus=false;
66                              this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                      }
68                      catch (SAXException e) // catch requerido para version 2.0
69                      {
70                              this.estatus=false;
71                              this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                      }
73
74              }
75
76              public CancelaOrden(String agenciaid, String password,
77                      String ordenid,  String cierre) {
78                      this.agenciaid=agenciaid;
79                      this.password=password;
80                      this.ordenid = ordenid;
81              }
82
83              public boolean ejecuta(ConnectionPool connectionPool) {
84                      String strSQL;
85                      // if requerido para version 2.0
86                      if (this.codigoError.equals( "5010-No hay datos para procesar"))
87                      {
88                              try {
89                                      Connection miconexion = connectionPool.getConnection();
90                                      miconexion.setAutoCommit(false);
91                                      String comilla ="'";
92                                      ResultSet rs ,rs2;
93                                      //String serieVehiculo;
94                                      PreparedStatement pstm;
95                                      Statement stm = miconexion.createStatement();
96                                      strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
```

```
97                                                  " WHERE agenciaid = " + comilla + agenciaid + comilla +
98                                                  " AND password = password("+comilla + password + comilla +")";
99                                  rs = stm.executeQuery(strSQL);
100                                 rs.next();
101                                 if (rs.getInt("conteo")!=1) //if 1
102                                 {
103                                         this.estatus=false;
104                                         this.codigoError="5000 - agenciaid:" + agenciaid +
105                                                         " con password:" + password + " no son validos. Contacte al
                                                                                                        administrador ";
106                                 } else {
107                                         strSQL = "SELECT count(*) as conteo FROM tblOrdenes " +
108                                                  " WHERE ordenid = " + comilla + this.ordenid + comilla +
109                                                  " AND agenciaid = " + comilla + this.agenciaid + comilla;
110                                                 rs=stm.executeQuery(strSQL);
111                                                 rs.next();
112                                                 if (rs.getInt("conteo")==0) { //if 2
113                                                         this.estatus=false;
114                                                          this.codigoError="5410 - La orden " + this.ordenid + " de la
                                                                                                        agencia "+this.agenciaid+
115                                                         " no existe. No es necesario eliminarla. ";
116                                                 } else {
117                                                         this.estatus=true;
118                                                         strSQL="DELETE FROM tblOrdenes " +
119                                                                 " WHERE ordenid=" + comilla + this.ordenid + comilla
                                                                                                                +
120                                                                         " AND agenciaid="  + comilla + this.agenciaid +
                                                                                                        comilla;
121                                                         pstm = miconexion.prepareStatement(strSQL);
122                                                         pstm.execute();
123                                                 } // end if 2
124                                 }// end if 1
125                                 rs.close();
126                                 miconexion.commit();
127                                 miconexion.close();
128                         } catch(SQLException sqle) {
129                           switch (sqle.getErrorCode())
130                           {
131                           case 0:
132                                          //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al
                                                                        administrador "+Integer.toString(inventario.size());
133                                 break;
134                           case 1049:
135                                 this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
136                                 break;
137                           case 1216: //para fks multiples se tendra que revisar que datos no son validos
138                                  this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                        administrador ";
139                                 break;
140                           default:
141                                  this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de
                                                                        datos. Contacte al administrador ";
142                           } // end switch
143                           //connectionPool = null;
144                           this.estatus=false;
145                         } //end catch
146                 }//end if
147                 return (this.estatus);
148         }
149
150
151         public SOAPMessage respuesta() {
152                 if (this.estatus)
153                 {
154                         return mensajeExitoso();
155                 } else {
156                         return mensajeError();
157                 }// end if
158         }
159
160         public SOAPMessage mensajeExitoso() {
161     SOAPMessage message = null;
162
163         try {
164             // create price list message
165             message = fac.createMessage();
166
167             // Access the SOAPBody object
168             SOAPPart part = message.getSOAPPart();
```

```
169                 SOAPEnvelope envelope = part.getEnvelope();
170                 SOAPBody body = envelope.getBody();
171                             SOAPHeader header = envelope.getHeader();
172                             header.detachNode();
173
174                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
175                     "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
176                 SOAPBodyElement list = body.addBodyElement(bodyName);
177
178                 Name exitosoN = envelope.createName("Respuesta");
179                 SOAPElement exitoso = list.addChildElement(exitosoN);
180                             exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
181                 message.saveChanges();
182
183             } catch(Exception e) {
184                 e.printStackTrace();
185             }
186             return message;
187         }// end mensajeExitoso
188
189             public SOAPMessage mensajeError() {
190         SOAPMessage message = null;
191
192             try {
193                 // create price list message
194                 message = fac.createMessage();
195
196
197                 // Access the SOAPBody object
198                 SOAPPart part = message.getSOAPPart();
199                 SOAPEnvelope envelope = part.getEnvelope();
200                 SOAPBody body = envelope.getBody();
201                             SOAPHeader header = envelope.getHeader();
202                             header.detachNode();
203
204                             SOAPFactory soapFactory = SOAPFactory.newInstance();
205                             SOAPFault fault = body.addFault();
206                             Name faultName =
207                                     soapFactory.createName("Server",
208                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
209                             fault.setFaultCode(faultName);
210                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
211                             fault.setFaultString(this.codigoError);
212                 message.saveChanges();
213
214             } catch(Exception e) {
215                 e.printStackTrace();
216             }
217             return message;
218         }// end servicioError
219
220
221             public String getAgenciaId() {
222                     return this.agenciaid;
223             }
224
225             public String getPassword() {
226                     return this.password;
227             }
228
229             public boolean getEstatus() {
230                     return this.estatus;
231             }
232
233             public String getCodigoError(){
234                     return this.codigoError;
235             }
236
237             public void setCodigoError(String codigoError) {
238                     this.codigoError=codigoError;
239             }
240
241     }
```

CancelaVenta.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
```

```
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
8     import javax.xml.soap.*;
9     import java.util.*;
10
11    // Bibliotecas para manejar la conexión de sql
12    import java.sql.*;
13    import coreservlets.*;
14
15    public class CancelaVenta
16    {
17            private String agenciaid="";
18            private String password="";
19            private String serieVehiculo="";
20            private boolean estatus = false;
21            private String codigoError = "5010-No hay datos para procesar";
22        static MessageFactory fac = null;
23
24        static {
25            try {
26                fac = MessageFactory.newInstance();
27            } catch (Exception ex) {
28                ex.printStackTrace();
29            }
30        }
31
32            public CancelaVenta(MimeHeaders headers, InputStream is)   throws IOException {
33                    try
34                    {
35                            SOAPMessage msg = fac.createMessage(headers, is);
36                            SOAPPart part = msg.getSOAPPart();
37
38                            // Código Agregado para la versión 2.0 del sistema
39                            // para validar datos desde la llegada del xml
40                            VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                    Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/CancelaVenta.xsd");
42                            Verifier verifier = schema.newVerifier();
43                            verifier.verify(part);
44                            // fin de código para version 2.0
45
46                            SOAPEnvelope envelope = part.getEnvelope();
47                            SOAPBody body = envelope.getBody();
48                            java.util.Iterator iterator = body.getChildElements();
49                            SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
50                            iterator = addUnit.getChildElements();
51                            SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                            SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                            SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
54                            this.agenciaid=agenciaid.getValue();
55                            this.password=password.getValue();
56                            this.serieVehiculo = serieVehiculo.getValue();
57                    }
58                    catch (SOAPException e)
59                    {
60                            this.estatus=false;
61                            this.codigoError="6000-Error al leer el xml de entrada en AltaUnidad";
62                    }
63                    catch (VerifierConfigurationException e) // catch requerido para version 2.0
64                    {
65                            this.estatus=false;
66                            this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
67                    }
68                    catch (SAXException e) // catch requerido para version 2.0
69                    {
70                            this.estatus=false;
71                            this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
72                    }
73            }
74
75            public CancelaVenta(String agenciaid, String password, String serieVehiculo) {
76                    this.agenciaid=agenciaid;
77                    this.password=password;
78                    this.serieVehiculo=serieVehiculo;
79            }
```

```
80
81          public boolean ejecuta(ConnectionPool connectionPool) {
82                  String strSQL;
83                  // if requerido para version 2.0
84                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
85                  {
86                          try {
87
88                                  Connection miconexion = connectionPool.getConnection();
89                                  String comilla ="'";
90                                  Statement stm = miconexion.createStatement();
91                                  strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
92                                                  "WHERE agenciaid = " + comilla + agenciaid + comilla +
93                                                  "AND password = password("+comilla + password + comilla +")";
94                                  ResultSet rs = stm.executeQuery(strSQL);
95                                  rs.next();
96                                  if (rs.getInt("conteo")!=1)
97                                  {
98                                          this.estatus=false;
99                                          this.codigoError="5000 - agenciaid:" + agenciaid +
100                                                          " con password:" + password + " no son validos. Contacte al
                                                                                                        administrador ";
101                                 } else {
102                                                 strSQL = "DELETE FROM tblVentas WHERE serievehiculo = " + comilla +
                                                                                                this.serieVehiculo + comilla +
103                                                         " AND agenciaid=" + comilla + this.agenciaid + comilla ;
104                                         PreparedStatement pstm = miconexion.prepareStatement(strSQL);
105                                         pstm.execute();
106                                         pstm.close();
107                                         this.estatus=true;
108                                 }
109                                 rs.close();
110                                 miconexion.close();
111                         } catch(SQLException sqle) {
112                           switch (sqle.getErrorCode())
113                           {
114                           case 0:
115                                           this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al
                                                                                                        administrador ";
116                                 break;
117                           case 1049:
118                                 this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
119                                 break;
120                           case 1062:
121                                          this.codigoError="1062 - Unidad " + this.serieVehiculo + " ya se encuentra en
                                                                                                        Inventario ";
122                                 break;
123                           case 1216: //para fks multiples se tendra que revisar que datos no son validos
124                                   this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo.
                                                                                                Contacte al administrador ";
125                                 break;
126                           default:
127                                  this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de
                                                                        datos. Contacte al administrador ";
128                           }// end switch
129                           //connectionPool = null;
130                           this.estatus=false;
131                         }// end catch
132                 }// end if
133                 return (this.estatus);
134         }
135
136
137         public SOAPMessage respuesta() {
138                 if (this.estatus)
139                 {
140                         return mensajeExitoso();
141                 } else {
142                         return mensajeError();
143                 }// end if
144         }
145
146         public SOAPMessage mensajeExitoso() {
147     SOAPMessage message = null;
148
149         try {
150             // create price list message
151             message = fac.createMessage();
152
```

```
153             // Access the SOAPBody object
154             SOAPPart part = message.getSOAPPart();
155             SOAPEnvelope envelope = part.getEnvelope();
156             SOAPBody body = envelope.getBody();
157                         SOAPHeader header = envelope.getHeader();
158                         header.detachNode();
159
160             Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
161                 "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
162             SOAPBodyElement list = body.addBodyElement(bodyName);
163
164             Name exitosoN = envelope.createName("Respuesta");
165             SOAPElement exitoso = list.addChildElement(exitosoN);
166                         exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
167             message.saveChanges();
168
169         } catch(Exception e) {
170             e.printStackTrace();
171         }
172         return message;
173     }// end mensajeExitoso
174
175         public SOAPMessage mensajeError() {
176     SOAPMessage message = null;
177
178         try {
179             // create price list message
180             message = fac.createMessage();
181
182
183             // Access the SOAPBody object
184             SOAPPart part = message.getSOAPPart();
185             SOAPEnvelope envelope = part.getEnvelope();
186             SOAPBody body = envelope.getBody();
187                         SOAPHeader header = envelope.getHeader();
188                         header.detachNode();
189
190                         SOAPFactory soapFactory = SOAPFactory.newInstance();
191                         SOAPFault fault = body.addFault();
192                         Name faultName =
193                                 soapFactory.createName("Server",
194                                 "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
195                         fault.setFaultCode(faultName);
196                          fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
197                         fault.setFaultString(this.codigoError);
198             message.saveChanges();
199
200         } catch(Exception e) {
201             e.printStackTrace();
202         }
203         return message;
204     }// end servicioError
205
206
207         public String getAgenciaId() {
208                 return this.agenciaid;
209         }
210
211         public String getPassword() {
212                 return this.password;
213         }
214
215         public String getSerieVehiculo() {
216                 return this.serieVehiculo;
217         }
218
219         public boolean getEstatus() {
220                 return this.estatus;
221         }
222
223         public String getCodigoError(){
224                 return this.codigoError;
225         }
226
227         public void setCodigoError(String codigoError) {
228                 this.codigoError=codigoError;
229         }
230
```

```
231     }
```

CierraOrden.java

```
1     // Bibliotecas para validar el mensaje XML SOAP
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
8     import javax.xml.soap.*;
9     import java.util.*;
10
11    // Bibliotecas para manejar la conexión de sql
12    import java.sql.*;
13    import coreservlets.*;
14
15    public class CierraOrden
16    {
17            private String agenciaid="";
18            private String password="";
19            private String ordenid="";
20            private String cierre="";
21            private boolean estatus = false;
22            private String codigoError = "5010-No hay datos para procesar";
23        static MessageFactory fac = null;
24
25        static {
26            try {
27                fac = MessageFactory.newInstance();
28            } catch (Exception ex) {
29                ex.printStackTrace();
30            }
31        }
32
33            public CierraOrden(MimeHeaders headers, InputStream is)    throws IOException {
34                    try
35                    {
36                            SOAPMessage msg = fac.createMessage(headers, is);
37                SOAPPart part = msg.getSOAPPart();
38
39                            // Código Agregado para la versión 2.0 del sistema
40                            // para validar datos desde la llegada del xml
41                            VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
42                                                                                                    Schema schema =
                                      factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/CierraOrden.xsd");
43                            Verifier verifier = schema.newVerifier();
44                            verifier.verify(part);
45                            // fin de código para version 2.0
46
47                            SOAPEnvelope envelope = part.getEnvelope();
48                            SOAPBody body = envelope.getBody();
49                            java.util.Iterator iterator = body.getChildElements();
50                            SOAPBodyElement cierraOrden = (SOAPBodyElement)iterator.next();
51                            iterator = cierraOrden.getChildElements();
52                            SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
53                            SOAPBodyElement password = (SOAPBodyElement)iterator.next();
54                            SOAPBodyElement ordenid = (SOAPBodyElement)iterator.next();
55                            SOAPBodyElement cierre = (SOAPBodyElement)iterator.next();
56                            this.agenciaid=agenciaid.getValue();
57                            this.password=password.getValue();
58                            this.ordenid = ordenid.getValue();
59                            this.cierre =  cierre.getValue();
60                    }
61                    catch (SOAPException e)
62                    {
63                            this.estatus=false;
64                            this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
65                    }
66                    catch (VerifierConfigurationException e) // catch requerido para version 2.0
67                    {
68                            this.estatus=false;
69                            this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
70                    }
71                    catch (SAXException e) // catch requerido para version 2.0
72                    {
73                            this.estatus=false;
```

```
74                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
75                  }
76
77          }
78
79          public CierraOrden(String agenciaid, String password,
80                  String ordenid,  String cierre) {
81                  this.agenciaid=agenciaid;
82                  this.password=password;
83                  this.ordenid = ordenid;
84                  this.cierre =  cierre;
85          }
86
87          public boolean ejecuta(ConnectionPool connectionPool) {
88                  String strSQL;
89                  // if requerido para version 2.0
90                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
91                  {
92                          try {
93                                  Connection miconexion = connectionPool.getConnection();
94                                  miconexion.setAutoCommit(false);
95                                  String comilla ="'";
96                                  ResultSet rs ,rs2;
97                                  //String serieVehiculo;
98                                  PreparedStatement pstm;
99                                  Statement stm = miconexion.createStatement();
100                                 strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
101                                                  " WHERE agenciaid = " + comilla + agenciaid + comilla +
102                                                  " AND password = password("+comilla + password + comilla +")";
103                                 rs = stm.executeQuery(strSQL);
104                                 rs.next();
105                                 if (rs.getInt("conteo")!=1) //if 1
106                                 {
107                                         this.estatus=false;
108                                         this.codigoError="5000 - agenciaid:" + agenciaid +
109                                                          " con password:" + password + " no son validos. Contacte al
                                                                                                  administrador ";
110                                 } else {
111                                         strSQL = "SELECT count(*) as conteo FROM tblOrdenes " +
112                                                  " WHERE ordenid = " + comilla + this.ordenid + comilla +
113                                                  " AND agenciaid = " + comilla + this.agenciaid + comilla;
114                                                 rs=stm.executeQuery(strSQL);
115                                                 rs.next();
116                                                 if (rs.getInt("conteo")==0) { //if 2
117                                                         this.estatus=false;
118                                                          this.codigoError="5420 - La orden " + this.ordenid + " de la
                                                                                                  agencia "+this.agenciaid+
119                                                         " no existe. No se le puede asignar una fecha de cierre ";
120                                                 } else {
121                                                         this.estatus=true;
122                                                         strSQL="UPDATE tblOrdenes SET " +
123                                                                 " cierre = " + comilla + this.cierre +comilla+
124                                                                 " WHERE ordenid=" + comilla + this.ordenid + comilla
                                                                                                                  +
125                                                                      " AND agenciaid="  + comilla + this.agenciaid +
                                                                                                          comilla;
126                                                         pstm = miconexion.prepareStatement(strSQL);
127                                                         pstm.execute();
128                                                 } // end if 2
129                                 }// end if 1
130                                 rs.close();
131                                 miconexion.commit();
132                                 miconexion.close();
133                         } catch(SQLException sqle) {
134                           switch (sqle.getErrorCode())
135                           {
136                           case 0:
137                                           //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al
                                                                  administrador "+Integer.toString(inventario.size());
138                                 break;
139                           case 1049:
140                                 this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
141                                 break;
142                           case 1216: //para fks multiples se tendra que revisar que datos no son validos
143                                  this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                  administrador ";
144                                 break;
145                           default:
146                                  this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de
```

```
                                                                    datos. Contacte al administrador ";
147                             } // end switch
148                             //connectionPool = null;
149                             this.estatus=false;
150                           } //end catch
151                   }//end if
152                   return (this.estatus);
153           }
154
155
156           public SOAPMessage respuesta() {
157                   if (this.estatus)
158                   {
159                           return mensajeExitoso();
160                   } else {
161                           return mensajeError();
162                   }// end if
163           }
164
165           public SOAPMessage mensajeExitoso() {
166       SOAPMessage message = null;
167
168           try {
169               // create price list message
170               message = fac.createMessage();
171
172               // Access the SOAPBody object
173               SOAPPart part = message.getSOAPPart();
174               SOAPEnvelope envelope = part.getEnvelope();
175               SOAPBody body = envelope.getBody();
176                           SOAPHeader header = envelope.getHeader();
177                           header.detachNode();
178
179               Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
180                   "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
181               SOAPBodyElement list = body.addBodyElement(bodyName);
182
183               Name exitosoN = envelope.createName("Respuesta");
184               SOAPElement exitoso = list.addChildElement(exitosoN);
185                           exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
186               message.saveChanges();
187
188           } catch(Exception e) {
189               e.printStackTrace();
190           }
191           return message;
192       }// end mensajeExitoso
193
194           public SOAPMessage mensajeError() {
195       SOAPMessage message = null;
196
197           try {
198               // create price list message
199               message = fac.createMessage();
200
201
202               // Access the SOAPBody object
203               SOAPPart part = message.getSOAPPart();
204               SOAPEnvelope envelope = part.getEnvelope();
205               SOAPBody body = envelope.getBody();
206                           SOAPHeader header = envelope.getHeader();
207                           header.detachNode();
208
209                           SOAPFactory soapFactory = SOAPFactory.newInstance();
210                           SOAPFault fault = body.addFault();
211                           Name faultName =
212                                   soapFactory.createName("Server",
213                                   "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
214                           fault.setFaultCode(faultName);
215                            fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
216                           fault.setFaultString(this.codigoError);
217               message.saveChanges();
218
219           } catch(Exception e) {
220               e.printStackTrace();
221           }
222           return message;
223       }// end servicioError
```

```
224
225
226             public String getAgenciaId() {
227                     return this.agenciaid;
228             }
229
230             public String getPassword() {
231                     return this.password;
232             }
233
234             public boolean getEstatus() {
235                     return this.estatus;
236             }
237
238             public String getCodigoError(){
239                     return this.codigoError;
240             }
241
242             public void setCodigoError(String codigoError) {
243                     this.codigoError=codigoError;
244             }
245
246     }
```

ImportesOrden.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class ImportesOrden
16      {
17              private String agenciaid="";
18              private String password="";
19              private String ordenid="";
20              private String ing_obratall="";
21              private String uti_obratall="";
22              private String ing_refatall="";
23              private String uti_refatall="";
24              private String ing_obrahyp="";
25              private String uti_obrahyp="";
26              private String ing_refahyp="";
27              private String uti_refahyp="";
28              private boolean estatus = false;
29              private String codigoError = "5010-No hay datos para procesar";
30          static MessageFactory fac = null;
31
32          static {
33              try {
34                  fac = MessageFactory.newInstance();
35              } catch (Exception ex) {
36                  ex.printStackTrace();
37              }
38          }
39
40              public ImportesOrden(MimeHeaders headers, InputStream is)  throws IOException {
41                      try
42                      {
43                              SOAPMessage msg = fac.createMessage(headers, is);
44                  SOAPPart part = msg.getSOAPPart();
45
46                              // Código Agregado para la versión 2.0 del sistema
47                              // para validar datos desde la llegada del xml
48                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
49                                                                                              Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/ImportesOrden.xsd");
50                              Verifier verifier = schema.newVerifier();
51                              verifier.verify(part);
```

```
52                          // fin de código para version 2.0
53
54                          SOAPEnvelope envelope = part.getEnvelope();
55              SOAPBody body = envelope.getBody();
56                          java.util.Iterator iterator = body.getChildElements();
57                          SOAPBodyElement importesorden = (SOAPBodyElement)iterator.next();
58                          iterator = importesorden.getChildElements();
59                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
60                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
61                          SOAPBodyElement ordenid = (SOAPBodyElement)iterator.next();
62                          SOAPBodyElement ing_obratall = (SOAPBodyElement)iterator.next();
63                          SOAPBodyElement uti_obratall = (SOAPBodyElement)iterator.next();
64                          SOAPBodyElement ing_refatall = (SOAPBodyElement)iterator.next();
65                          SOAPBodyElement uti_refatall = (SOAPBodyElement)iterator.next();
66                          SOAPBodyElement ing_obrahyp = (SOAPBodyElement)iterator.next();
67                          SOAPBodyElement uti_obrahyp = (SOAPBodyElement)iterator.next();
68                          SOAPBodyElement ing_refahyp = (SOAPBodyElement)iterator.next();
69                          SOAPBodyElement uti_refahyp = (SOAPBodyElement)iterator.next();
70                          this.agenciaid=agenciaid.getValue();
71                          this.password=password.getValue();
72                          this.ordenid = ordenid.getValue();
73                          this.ing_obratall =  ing_obratall.getValue();
74                          this.uti_obratall =  uti_obratall.getValue();
75                          this.ing_refatall =  ing_refatall.getValue();
76                          this.uti_refatall =  uti_refatall.getValue();
77                          this.ing_obrahyp  =  ing_obrahyp.getValue();
78                          this.uti_obrahyp  =  uti_obrahyp.getValue();
79                          this.ing_refahyp  =  ing_refahyp.getValue();
80                          this.uti_refahyp  =  uti_refahyp.getValue();
81                  }
82                  catch (SOAPException e)
83                  {
84                          this.estatus=false;
85                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
86                  }
87                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
88                  {
89                          this.estatus=false;
90                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
91                  }
92                  catch (SAXException e) // catch requerido para version 2.0
93                  {
94                          this.estatus=false;
95                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
96                  }
97          }
98
99          public ImportesOrden(String agenciaid, String password, String ordenid,
100                 String ing_obratall, String uti_obratall, String ing_refatall,
101                 String uti_refatall, String ing_obrahyp, String uti_obrahyp,
102                 String ing_refahyp, String uti_refahyp) {
103                 this.agenciaid=agenciaid;
104                 this.password=password;
105                 this.ordenid = ordenid;
106                 this.ing_obratall =  ing_obratall;
107                 this.uti_obratall =  uti_obratall;
108                 this.ing_refatall =  ing_refatall;
109                 this.uti_refatall =  uti_refatall;
110                 this.ing_obrahyp  =  ing_obrahyp;
111                 this.uti_obrahyp  =  uti_obrahyp;
112                 this.ing_refahyp  =  ing_refahyp;
113                 this.uti_refahyp  =  uti_refahyp;
114         }
115
116         public boolean ejecuta(ConnectionPool connectionPool) {
117                 String strSQL;
118                 // if requerido para version 2.0
119                 if (this.codigoError.equals( "5010-No hay datos para procesar"))
120                 {
121                 try {
122                         Connection miconexion = connectionPool.getConnection();
123                         miconexion.setAutoCommit(false);
124                         String comilla ="'";
125                         ResultSet rs ,rs2;
126                         //String serieVehiculo;
127                         PreparedStatement pstm;
128                         Statement stm = miconexion.createStatement();
129                         strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
130                                 " WHERE agenciaid = " + comilla + agenciaid + comilla +
```

```
131                                     " AND password = password("+comilla + password + comilla +")";
132                             rs = stm.executeQuery(strSQL);
133                             rs.next();
134                             if (rs.getInt("conteo")!=1) //if 1
135                             {
136                                     this.estatus=false;
137                                 this.codigoError="5000 - agenciaid:" + agenciaid +
138                                             " con password:" + password + " no son validos. Contacte al administrador ";
139                             } else {
140                                     strSQL = "SELECT count(*) as conteo FROM tblOrdenes " +
141                                              " WHERE ordenid = " + comilla + this.ordenid + comilla +
142                                              " AND agenciaid = " + comilla + this.agenciaid + comilla;
143                                             rs=stm.executeQuery(strSQL);
144                                             rs.next();
145                                             if (rs.getInt("conteo")==0) { //if 2
146                                                     this.estatus=false;
147                                                      this.codigoError="5400 - La orden " + this.ordenid + " de la agencia
                                                                                                                  "+this.agenciaid+
148                                                     " no existe. No pueden ser actualizados los importes ";
149                                             } else {
150                                                     this.estatus=true;
151                                                     strSQL="UPDATE tblOrdenes SET " +
152                                                             " ing_obratall = " + this.ing_obratall + "," +
153                                                             " uti_obratall = " + this.uti_obratall + "," +
154                                                             " ing_refatall = " + this.ing_refatall + "," +
155                                                             " uti_refatall = " + this.uti_refatall + "," +
156                                                             " ing_obrahyp = " + this.ing_obrahyp + "," +
157                                                             " uti_obrahyp = " + this.uti_obrahyp + "," +
158                                                             " ing_refahyp = " + this.ing_refahyp + "," +
159                                                             " uti_refahyp = " + this.uti_refahyp +
160                                                             " WHERE ordenid="+comilla+this.ordenid+comilla +
161                                                             " AND agenciaid=" +comilla+this.agenciaid+comilla;
162                                                     pstm = miconexion.prepareStatement(strSQL);
163                                                     pstm.execute();
164                                             } // end if 2
165                             }// end if 1
166                             rs.close();
167                             miconexion.commit();
168                             miconexion.close();
169                     } catch(SQLException sqle) {
170                       switch (sqle.getErrorCode())
171                       {
172                       case 0:
173                                 //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                      "+Integer.toString(inventario.size());
174                             break;
175                       case 1049:
176                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
177                             break;
178                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
179                                       this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                                  administrador ";
180                             break;
181                       default:
182                                this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                  Contacte al administrador ";
183                       } // end switch
184                       //connectionPool = null;
185                       this.estatus=false;
186                     } // end catch
187                     }// end if
188                     return (this.estatus);
189             }
190
191
192             public SOAPMessage respuesta() {
193                     if (this.estatus)
194                     {
195                             return mensajeExitoso();
196                     } else {
197                             return mensajeError();
198                     }// end if
199             }
200
201             public SOAPMessage mensajeExitoso() {
202             SOAPMessage message = null;
203
204             try {
205                 // create price list message
```

```
206                 message = fac.createMessage();
207
208                 // Access the SOAPBody object
209                 SOAPPart part = message.getSOAPPart();
210                 SOAPEnvelope envelope = part.getEnvelope();
211                 SOAPBody body = envelope.getBody();
212                             SOAPHeader header = envelope.getHeader();
213                             header.detachNode();
214
215                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
216                     "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
217                 SOAPBodyElement list = body.addBodyElement(bodyName);
218
219                 Name exitosoN = envelope.createName("Respuesta");
220                 SOAPElement exitoso = list.addChildElement(exitosoN);
221                             exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
222                 message.saveChanges();
223
224             } catch(Exception e) {
225                 e.printStackTrace();
226             }
227             return message;
228         }// end mensajeExitoso
229
230             public SOAPMessage mensajeError() {
231         SOAPMessage message = null;
232
233             try {
234                 // create price list message
235                 message = fac.createMessage();
236
237
238                 // Access the SOAPBody object
239                 SOAPPart part = message.getSOAPPart();
240                 SOAPEnvelope envelope = part.getEnvelope();
241                 SOAPBody body = envelope.getBody();
242                             SOAPHeader header = envelope.getHeader();
243                             header.detachNode();
244
245                             SOAPFactory soapFactory = SOAPFactory.newInstance();
246                             SOAPFault fault = body.addFault();
247                             Name faultName =
248                                     soapFactory.createName("Server",
249                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
250                             fault.setFaultCode(faultName);
251                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
252                             fault.setFaultString(this.codigoError);
253                 message.saveChanges();
254
255             } catch(Exception e) {
256                 e.printStackTrace();
257             }
258             return message;
259         }// end servicioError
260
261
262             public String getAgenciaId() {
263                     return this.agenciaid;
264             }
265
266             public String getPassword() {
267                     return this.password;
268             }
269
270             public boolean getEstatus() {
271                     return this.estatus;
272             }
273
274             public String getCodigoError(){
275                     return this.codigoError;
276             }
277
278             public void setCodigoError(String codigoError) {
279                     this.codigoError=codigoError;
280             }
281
282     }
```

InstanciaSoap.java

```
1   import java.io.*;
2   import javax.servlet.*;
3   import javax.servlet.http.*;
4   import javax.xml.soap.*;
5   import java.util.*;
6   import java.sql.*;
7   import coreservlets.*;
8
9   /* Utilería para leer XML SOAP de los clientes
10  */
11  public class InstanciaSOAP extends HttpServlet {
12          FileOutputStream fos;
13          OutputStreamWriter osw;
14          public ConnectionPool connectionPool;
15      static MessageFactory fac = null;
16
17      static {
18          try {
19              fac = MessageFactory.newInstance();
20          } catch (Exception ex) {
21              ex.printStackTrace();
22          }
23      }
24
25
26          public void doGet(HttpServletRequest request, HttpServletResponse response)
27      throws IOException, ServletException
28      {
29                  doPost(request,response);
30          }
31
32      public void doPost(HttpServletRequest request, HttpServletResponse response)
33      throws IOException, ServletException
34      {
35                  try
36                  {
37                          //=============================================================================================
38                          // leer el nombre del metodo y los parametros del SOAPRequest
39                          //=============================================================================================
40
41                          // Obtener solamente el nombre del metodo RPC de los headers
42                          String soapActionString = request.getHeader("soapaction");
43                          soapActionString=soapActionString.substring(1,soapActionString.length()-1);
44                          //osw.write("soapaction\n"+soapActionString+"\n");
45
46                          // Get all the headers from the HTTP request
47                          MimeHeaders headers = getHeaders(request);
48
49                          // Get the body of the HTTP request
50                          InputStream is = request.getInputStream();
51                          SOAPMessage msg = fac.createMessage(headers, is);
52
53
54                          // Preparar la respuesta para todos lo metodos
55                          response.setHeader("Content-Type", "text/xml; charset=utf-8");
56                          javax.servlet.ServletOutputStream sos = response.getOutputStream();
57                          FileOutputStream instanciaSoap = new FileOutputStream("InstanciaSOAP.XML");
58                          msg.writeTo(sos);
59                          msg.writeTo(instanciaSoap);
60                          sos.close();
61                          instanciaSoap.close();
62
63                  }
64                  catch (javax.xml.soap.SOAPException ex)           {               }
65
66      }//end doPost
67
68      static MimeHeaders getHeaders(HttpServletRequest req) {
69
70          Enumeration enum = req.getHeaderNames();
71          MimeHeaders headers = new MimeHeaders();
72
73          while (enum.hasMoreElements()) {
74              String headerName = (String)enum.nextElement();
75              String headerValue = req.getHeader(headerName);
76
```

```
77                StringTokenizer values =
78                    new StringTokenizer(headerValue, ",");
79                while (values.hasMoreTokens()) {
80                    headers.addHeader(headerName,
81                        values.nextToken().trim());
82                }
83            }
84            return headers;
85        }//end getHeaders
86
87        static void putHeaders(MimeHeaders headers,
88                HttpServletResponse res) {
89
90            Iterator it = headers.getAllHeaders();
91            while (it.hasNext()) {
92                MimeHeader header = (MimeHeader)it.next();
93
94                String[] values = headers.getHeader(header.getName());
95                if (values.length == 1) {
96                    res.setHeader(header.getName(), header.getValue());
97                } else {
98                    StringBuffer concat = new StringBuffer();
99                    int i = 0;
100                   while (i < values.length) {
101                       if (i != 0) {
102                           concat.append(',');
103                       }
104                       concat.append(values[i++]);
105                   }
106                   res.setHeader(header.getName(), concat.toString());
107               }
108           }
109       }//end putHeaders
110
111           static void escribeHeaders(MimeHeaders headers, OutputStreamWriter osw)
112                    throws java.io.IOException
113                    {
114                    //esta rutina es un clon de putHeaders, la utilize solo para ver que headers trae
115                    //el soap request. El header que me interesa es el soapaction porque me dice que
116                    //metodo web tiene que ejecutar el servicio.
117           Iterator it = headers.getAllHeaders();
118           while (it.hasNext()) {
119               MimeHeader header = (MimeHeader)it.next();
120
121               String[] values = headers.getHeader(header.getName());
122               if (values.length == 1) {
123                   osw.write(header.getName()+"\n");
124                                   osw.write(header.getValue()+"\n");
125                                   osw.write("---"+"\n");
126               } else {
127                   StringBuffer concat = new StringBuffer();
128                   int i = 0;
129                   while (i < values.length) {
130                       if (i != 0) {
131                           concat.append(',');
132                       }
133                       concat.append(values[i++]);
134                   }
135                   osw.write(header.getName()+"\n");
136                                   osw.write(concat.toString()+"\n");
137                                   osw.write("---"+"\n");
138               }
139                    }//end while
140           }//end escribeHeaders
141   }
```

ListaAsesores.java

```
1     // Bibliotecas para validar el mensaje XML SOAP
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
8     import javax.xml.soap.*;
9     import java.util.*;
```

```
10
11    // Bibliotecas para manejar la conexión de sql
12    import java.sql.*;
13    import coreservlets.*;
14
15    public class ListaAsesores
16    {
17            private String agenciaid="";
18            private String password="";
19            private SOAPMessage islaDatosXML;
20            private boolean estatus = false;
21            private String codigoError = "5010-No hay datos para procesar";
22
23            static MessageFactory fac = null;
24            static {
25            try {
26                    fac = MessageFactory.newInstance();
27            } catch (Exception ex) {
28               ex.printStackTrace();
29            }
30            }
31
32            public ListaAsesores(MimeHeaders headers, InputStream is)  throws IOException {
33                    try
34                    {
35                            SOAPMessage msg = fac.createMessage(headers, is);
36                       SOAPPart part = msg.getSOAPPart();
37
38                            // Código Agregado para la versión 2.0 del sistema
39                            // para validar datos desde la llegada del xml
40                            VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                    Schema schema =
                                      factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/ListaAsesores.xsd");
42                            Verifier verifier = schema.newVerifier();
43                            verifier.verify(part);
44                            // fin de código para version 2.0
45
46                            SOAPEnvelope envelope = part.getEnvelope();
47                            SOAPBody body = envelope.getBody();
48                            java.util.Iterator iterator = body.getChildElements();
49                            SOAPBodyElement listaAsesores = (SOAPBodyElement)iterator.next();
50                            iterator = listaAsesores.getChildElements();
51                            SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                            SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                            this.agenciaid=agenciaid.getValue();
54                            this.password=password.getValue();
55                    }
56                    catch (SOAPException e)
57                    {
58                            this.estatus=false;
59                            this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
60                    }
61                    catch (VerifierConfigurationException e) // catch requerido para version 2.0
62                    {
63                            this.estatus=false;
64                            this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
65                    }
66                    catch (SAXException e) // catch requerido para version 2.0
67                    {
68                            this.estatus=false;
69                            this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
70                    }
71
72            }
73
74            public ListaAsesores(String agenciaid, String password) {
75                    this.agenciaid=agenciaid;
76                    this.password=password;
77            }
78
79            public boolean ejecuta(ConnectionPool connectionPool) {
80                    String strSQL;
81                    // if requerido para version 2.0
82                    if (this.codigoError.equals( "5010-No hay datos para procesar"))
83                    {
84                    try {
85                            Connection miconexion = connectionPool.getConnection();
86                            String comilla ="'";
87                            ResultSet rs ,rs2;
```

```
88                              //String serieVehiculo;
89                              PreparedStatement pstm;
90                              Statement stm = miconexion.createStatement();
91                              strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
92                                       " WHERE agenciaid = " + comilla + agenciaid + comilla +
93                                            " AND password = password("+comilla + password + comilla +")";
94                              rs = stm.executeQuery(strSQL);
95                              rs.next();
96                              if (rs.getInt("conteo")!=1) //if 1
97                              {
98                                      this.estatus=false;
99                                  this.codigoError="5000 - agenciaid:" + agenciaid +
100                                             " con password:" + password + " no son validos. Contacte al administrador ";
101                             } else {
102                                     this.estatus=true;
103                                     this.islaDatosXML =fac.createMessage();
104                                     SOAPPart part = this.islaDatosXML.getSOAPPart();
105                                     SOAPEnvelope envelope = part.getEnvelope();
106                                     SOAPBody body = envelope.getBody();
107                                     SOAPHeader header = envelope.getHeader();
108                                     header.detachNode();
109
110                                     Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
111                                             "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
112                                     SOAPBodyElement respuesta = body.addBodyElement(bodyName);
113
114                                     strSQL = "SELECT rfcAsesor, nombre "+
115                                                     " FROM tblAsesores " +
116                                                     " WHERE agenciaid= " + comilla + this.agenciaid + comilla;
117                                     rs = stm.executeQuery(strSQL);
118                                     while (rs.next())
119                                     {
120                                             Name asesorN = envelope.createName("asesor");
121                                             SOAPElement asesor = respuesta.addChildElement(asesorN);
122
123                                             Name rfcAsesorN = envelope.createName("rfcAsesor");
124                                             SOAPElement rfcAsesor = asesor.addChildElement(rfcAsesorN);
125                                             rfcAsesor.addTextNode(rs.getString("rfcAsesor"));
126
127                                             Name nombreN = envelope.createName("nombre");
128                                             SOAPElement nombre = asesor.addChildElement(nombreN);
129                                             nombre.addTextNode(rs.getString("nombre"));
130                                     }//end while rs1
131                                     this.islaDatosXML.saveChanges();
132
133                                     }// end if 1
134                             rs.close();
135                             miconexion.close();
136                     } catch(SQLException sqle) {
137                       switch (sqle.getErrorCode())
138                       {
139                       case 0:
140                                 //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                "+Integer.toString(inventario.size());
141                             break;
142                       case 1049:
143                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
144                             break;
145                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
146                                       this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                        administrador ";
147                             break;
148                       default:
149                                this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                        Contacte al administrador ";
150                       } // end switch
151                       //connectionPool = null;
152                       this.estatus=false;
153                     } catch(SOAPException e) {
154                 e.printStackTrace();
155             } // end catch
156                     } //end if
157                     return (this.estatus);
158             }
159
160
161             public SOAPMessage respuesta() {
162                     if (this.estatus)
163                     {
```

```
164                             return this.islaDatosXML;
165                     } else {
166                             return mensajeError();
167                     }// end if
168             }
169
170             public SOAPMessage mensajeError() {
171             SOAPMessage message = null;
172
173             try {
174                 // create price list message
175                 message = fac.createMessage();
176
177
178                 // Access the SOAPBody object
179                 SOAPPart part = message.getSOAPPart();
180                 SOAPEnvelope envelope = part.getEnvelope();
181                 SOAPBody body = envelope.getBody();
182                             SOAPHeader header = envelope.getHeader();
183                             header.detachNode();
184
185                             SOAPFactory soapFactory = SOAPFactory.newInstance();
186                             SOAPFault fault = body.addFault();
187                             Name faultName =
188                                     soapFactory.createName("Server",
189                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
190                             fault.setFaultCode(faultName);
191                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
192                             fault.setFaultString(this.codigoError);
193                 message.saveChanges();
194
195             } catch(Exception e) {
196                 e.printStackTrace();
197             }
198             return message;
199         }// end servicioError
200
201
202             public String getAgenciaId() {
203                     return this.agenciaid;
204             }
205
206             public String getPassword() {
207                     return this.password;
208             }
209
210             public boolean getEstatus() {
211                     return this.estatus;
212             }
213
214             public String getCodigoError(){
215                     return this.codigoError;
216             }
217
218             public void setCodigoError(String codigoError) {
219                     this.codigoError=codigoError;
220             }
221
222     }
```

ListaClientes.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
```

```
15  public class ListaClientes
16  {
17          private String agenciaid="";
18          private String password="";
19          private String rfc_o_curp_cliente="";
20          private SOAPMessage islaDatosXML;
21          private boolean estatus = false;
22          private String codigoError = "5010-No hay datos para procesar";
23
24          static MessageFactory fac = null;
25      static {
26          try {
27              fac = MessageFactory.newInstance();
28          } catch (Exception ex) {
29              ex.printStackTrace();
30          }
31      }
32
33          public ListaClientes(MimeHeaders headers, InputStream is)  throws IOException {
34                  try
35                  {
36                          SOAPMessage msg = fac.createMessage(headers, is);
37              SOAPPart part = msg.getSOAPPart();
38
39                          // Código Agregado para la versión 2.0 del sistema
40                          // para validar datos desde la llegada del xml
41                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
42                                                                                                  Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/ListaClientes.xsd");
43                          Verifier verifier = schema.newVerifier();
44                          verifier.verify(part);
45                          // fin de código para version 2.0
46
47                          SOAPEnvelope envelope = part.getEnvelope();
48                          SOAPBody body = envelope.getBody();
49                          java.util.Iterator iterator = body.getChildElements();
50                          SOAPBodyElement cierraOrden = (SOAPBodyElement)iterator.next();
51                          iterator = cierraOrden.getChildElements();
52                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
53                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
54                          this.agenciaid=agenciaid.getValue();
55                          this.password=password.getValue();
56                  }
57                  catch (SOAPException e)
58                  {
59                          this.estatus=false;
60                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
61                  }
62                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
63                  {
64                          this.estatus=false;
65                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
66                  }
67                  catch (SAXException e) // catch requerido para version 2.0
68                  {
69                          this.estatus=false;
70                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
71                  }
72
73          }
74
75          public ListaClientes(String agenciaid, String password) {
76                  this.agenciaid=agenciaid;
77                  this.password=password;
78          }
79
80          public boolean ejecuta(ConnectionPool connectionPool) {
81                  String strSQL;
82                  // if requerido para version 2.0
83                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
84                  {
85                  try {
86                          Connection miconexion = connectionPool.getConnection();
87                          String comilla ="'";
88                          ResultSet rs ,rs2;
89                          //String serieVehiculo;
90                          PreparedStatement pstm;
91                          Statement stm = miconexion.createStatement();
92                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
```

```
93                                   " WHERE agenciaid = " + comilla + agenciaid + comilla +
94                                       " AND password = password("+comilla + password + comilla +")";
95                              rs = stm.executeQuery(strSQL);
96                              rs.next();
97                              if (rs.getInt("conteo")!=1) //if 1
98                              {
99                                      this.estatus=false;
100                                 this.codigoError="5000 - agenciaid:" + agenciaid +
101                                              " con password:" + password + " no son validos. Contacte al administrador ";
102                             } else {
103                                     this.estatus=true;
104                                     this.islaDatosXML =fac.createMessage();
105                                     SOAPPart part = this.islaDatosXML.getSOAPPart();
106                         SOAPEnvelope envelope = part.getEnvelope();
107                             SOAPBody body = envelope.getBody();
108                                     SOAPHeader header = envelope.getHeader();
109                                     header.detachNode();
110
111                                     Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
112                                             "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
113                                     SOAPBodyElement respuesta = body.addBodyElement(bodyName);
114
115                                     strSQL = "SELECT rfc_o_curp_cliente, estado, clave, nombre, paterno,"+
116                                                     "materno, razonsocial, calle, numero, colonia, municipio, cp"+
117                                                     " FROM tblClientes " +
118                                                     " WHERE agenciaid= " + comilla + this.agenciaid + comilla;
119                                     rs = stm.executeQuery(strSQL);
120                                     while (rs.next())
121                                     {
122
123                                     Name clienteN = envelope.createName("cliente");
124                                     SOAPElement cliente = respuesta.addChildElement(clienteN);
125
126                                     Name rfc_o_curp_clienteN = envelope.createName("rfc_o_curp_cliente");
127                                     SOAPElement rfc_o_curp_cliente = cliente.addChildElement(rfc_o_curp_clienteN);
128                                     rfc_o_curp_cliente.addTextNode(rs.getString("rfc_o_curp_cliente"));
129
130                                     Name estadoN = envelope.createName("estado");
131                                     SOAPElement estado = cliente.addChildElement(estadoN);
132                                     estado.addTextNode(rs.getString("estado"));
133
134                                     Name claveN = envelope.createName("clave");
135                                     SOAPElement clave = cliente.addChildElement(claveN);
136                                     clave.addTextNode(rs.getString("clave"));
137
138                                     Name nombreN = envelope.createName("nombre");
139                                     SOAPElement nombre = cliente.addChildElement(nombreN);
140                                     nombre.addTextNode(rs.getString("nombre"));
141
142                                     Name paternoN = envelope.createName("paterno");
143                                     SOAPElement paterno = cliente.addChildElement(paternoN);
144                                     paterno.addTextNode(rs.getString("paterno"));
145
146                                     Name maternoN = envelope.createName("materno");
147                                     SOAPElement materno = cliente.addChildElement(maternoN);
148                                     materno.addTextNode(rs.getString("materno"));
149
150                                     Name razonsocialN = envelope.createName("razonsocial");
151                                     SOAPElement razonsocial = cliente.addChildElement(razonsocialN);
152                                     razonsocial.addTextNode(rs.getString("razonsocial"));
153
154                                     Name calleN = envelope.createName("calle");
155                                     SOAPElement calle = cliente.addChildElement(calleN);
156                                     calle.addTextNode(rs.getString("calle"));
157
158                                     Name numerocalleN = envelope.createName("numero");
159                                     SOAPElement numerocalle= cliente.addChildElement(numerocalleN);
160                                     numerocalle.addTextNode(rs.getString("numero"));
161
162                                     Name coloniaN = envelope.createName("colonia");
163                                     SOAPElement colonia = cliente.addChildElement(coloniaN);
164                                     colonia.addTextNode(rs.getString("colonia"));
165
166                                     Name municipioN = envelope.createName("municipio");
167                                     SOAPElement municipio = cliente.addChildElement(municipioN);
168                                     municipio.addTextNode(rs.getString("municipio"));
169
170                                     Name cpN = envelope.createName("cp");
171                                     SOAPElement cp = cliente.addChildElement(cpN);
```

```
172                                 cp.addTextNode(rs.getString("cp"));
173
174                                 Name telefonosN = envelope.createName("telefonos");
175                                 SOAPElement telefonos = cliente.addChildElement(telefonosN);
176
177
178                                 strSQL = "SELECT tipodetelefono, lada, telefono "+
179                                                 " FROM tblTelefonos " +
180                                                                                 " WHERE rfc_o_curp_cliente= " + comilla +
                                                                                rs.getString("rfc_o_curp_cliente") + comilla+
181                                                 " AND agenciaid = " + comilla + this.agenciaid + comilla;
182                                 Statement stm2 = miconexion.createStatement();
183                                 rs2 = stm2.executeQuery(strSQL);
184                                 while (rs2.next()) {
185                                         Name telefonoN = envelope.createName("telefono");
186                                         SOAPElement telefono = telefonos.addChildElement(telefonoN);
187
188                                                 Name tipodetelefonoN = envelope.createName("tipodetelefono");
189                                                                                         SOAPElement tipodetelefono =
                                                                                    telefono.addChildElement(tipodetelefonoN);
190                                                 tipodetelefono.addTextNode(rs2.getString("tipodetelefono"));
191
192                                                 Name ladaN = envelope.createName("lada");
193                                                 SOAPElement lada = telefono.addChildElement(ladaN);
194                                                 lada.addTextNode(rs2.getString("lada"));
195
196                                                 Name numeroN = envelope.createName("numero");
197                                                 SOAPElement numero = telefono.addChildElement(numeroN);
198                                                 numero.addTextNode(rs2.getString("telefono"));
199                                 }//end while rs2
200                                 }//end while rs1
201                                 this.islaDatosXML.saveChanges();
202
203                                 }// end if 1
204                         rs.close();
205                         miconexion.close();
206                 } catch(SQLException sqle) {
207                   switch (sqle.getErrorCode())
208                   {
209                   case 0:
210                             //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                "+Integer.toString(inventario.size());
211                         break;
212                   case 1049:
213                     this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
214                         break;
215                   case 1216: //para fks multiples se tendra que revisar que datos no son validos
216                                 this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                        administrador ";
217                         break;
218                   default:
219                            this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                        Contacte al administrador ";
220                   } // end switch
221                   //connectionPool = null;
222                   this.estatus=false;
223                 } catch(SOAPException e) {
224             e.printStackTrace();
225           } // end catch
226                 }// end if
227                 return (this.estatus);
228           }
229
230
231           public SOAPMessage respuesta() {
232                 if (this.estatus)
233                 {
234                         return this.islaDatosXML;
235                 } else {
236                         return mensajeError();
237                 }// end if
238           }
239
240           public SOAPMessage mensajeError() {
241           SOAPMessage message = null;
242
243           try {
244               // create price list message
245               message = fac.createMessage();
```

```
246
247
248                 // Access the SOAPBody object
249                 SOAPPart part = message.getSOAPPart();
250                 SOAPEnvelope envelope = part.getEnvelope();
251                 SOAPBody body = envelope.getBody();
252                             SOAPHeader header = envelope.getHeader();
253                             header.detachNode();
254
255                             SOAPFactory soapFactory = SOAPFactory.newInstance();
256                             SOAPFault fault = body.addFault();
257                             Name faultName =
258                                     soapFactory.createName("Server",
259                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
260                             fault.setFaultCode(faultName);
261                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
262                             fault.setFaultString(this.codigoError);
263                 message.saveChanges();
264
265             } catch(Exception e) {
266                 e.printStackTrace();
267             }
268             return message;
269         }// end servicioError
270
271
272             public String getAgenciaId() {
273                     return this.agenciaid;
274             }
275
276             public String getPassword() {
277                     return this.password;
278             }
279
280             public boolean getEstatus() {
281                     return this.estatus;
282             }
283
284             public String getCodigoError(){
285                     return this.codigoError;
286             }
287
288             public void setCodigoError(String codigoError) {
289                     this.codigoError=codigoError;
290             }
291
292     }
```

ListaVendedores.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class ListaVendedores
16      {
17              private String agenciaid="";
18              private String password="";
19              private SOAPMessage islaDatosXML;
20              private boolean estatus = false;
21              private String codigoError = "5010-No hay datos para procesar";
22
23              static MessageFactory fac = null;
24          static {
25              try {
```

```
26              fac = MessageFactory.newInstance();
27          } catch (Exception ex) {
28              ex.printStackTrace();
29          }
30      }
31
32          public ListaVendedores(MimeHeaders headers, InputStream is)        throws IOException {
33                  try
34                  {
35                          SOAPMessage msg = fac.createMessage(headers, is);
36                          SOAPPart part = msg.getSOAPPart();
37
38                          // Código Agregado para la versión 2.0 del sistema
39                          // para validar datos desde la llegada del xml
40                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
41                                                                                                  Schema schema =
                                factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/ListaVendedores.xsd");
42                          Verifier verifier = schema.newVerifier();
43                          verifier.verify(part);
44                          // fin de código para version 2.0
45
46                          SOAPEnvelope envelope = part.getEnvelope();
47                          SOAPBody body = envelope.getBody();
48                          java.util.Iterator iterator = body.getChildElements();
49                          SOAPBodyElement listaVendedores = (SOAPBodyElement)iterator.next();
50                          iterator = listaVendedores.getChildElements();
51                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
52                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
53                          this.agenciaid=agenciaid.getValue();
54                          this.password=password.getValue();
55                  }
56                  catch (SOAPException e)
57                  {
58                          this.estatus=false;
59                          this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
60                  }
61                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
62                  {
63                          this.estatus=false;
64                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
65                  }
66                  catch (SAXException e) // catch requerido para version 2.0
67                  {
68                          this.estatus=false;
69                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
70                  }
71
72          }
73
74          public ListaVendedores(String agenciaid, String password) {
75                  this.agenciaid=agenciaid;
76                  this.password=password;
77          }
78
79          public boolean ejecuta(ConnectionPool connectionPool) {
80                  String strSQL;
81                  // if requerido para version 2.0
82                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
83                  {
84                  try {
85                          Connection miconexion = connectionPool.getConnection();
86                          String comilla ="'";
87                          ResultSet rs ,rs2;
88                          //String serieVehiculo;
89                          PreparedStatement pstm;
90                          Statement stm = miconexion.createStatement();
91                          strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
92                                   " WHERE agenciaid = " + comilla + agenciaid + comilla +
93                                       " AND password = password("+comilla + password + comilla +")";
94                          rs = stm.executeQuery(strSQL);
95                          rs.next();
96                          if (rs.getInt("conteo")!=1) //if 1
97                          {
98                                  this.estatus=false;
99                              this.codigoError="5000 - agenciaid:" + agenciaid +
100                                         " con password:" + password + " no son validos. Contacte al administrador ";
101                         } else {
102                                 this.estatus=true;
103                                 this.islaDatosXML =fac.createMessage();
```

```
104                                 SOAPPart part = this.islaDatosXML.getSOAPPart();
105                         SOAPEnvelope envelope = part.getEnvelope();
106                             SOAPBody body = envelope.getBody();
107                                 SOAPHeader header = envelope.getHeader();
108                                 header.detachNode();
109
110                                 Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
111                                         "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
112                                 SOAPBodyElement respuesta = body.addBodyElement(bodyName);
113
114                                 strSQL = "SELECT rfcVendedor, nombre "+
115                                                 " FROM tblVendedores " +
116                                                 " WHERE agenciaid= " + comilla + this.agenciaid + comilla;
117                                 rs = stm.executeQuery(strSQL);
118                                 while (rs.next())
119                                 {
120                                         Name vendedorN = envelope.createName("vendedor");
121                                         SOAPElement vendedor = respuesta.addChildElement(vendedorN);
122
123                                         Name rfcVendedorN = envelope.createName("rfcVendedor");
124                                         SOAPElement rfcVendedor = vendedor.addChildElement(rfcVendedorN);
125                                         rfcVendedor.addTextNode(rs.getString("rfcVendedor"));
126
127                                         Name nombreN = envelope.createName("nombre");
128                                         SOAPElement nombre = vendedor.addChildElement(nombreN);
129                                         nombre.addTextNode(rs.getString("nombre"));
130                                 }//end while rs1
131                                 this.islaDatosXML.saveChanges();
132
133                                 }// end if 1
134                             rs.close();
135                             miconexion.close();
136                     } catch(SQLException sqle) {
137                       switch (sqle.getErrorCode())
138                       {
139                       case 0:
140                                 //this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador
                                                                                                    "+Integer.toString(inventario.size());
141                             break;
142                       case 1049:
143                         this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
144                             break;
145                       case 1216: //para fks multiples se tendra que revisar que datos no son validos
146                                     this.codigoError="1216 - Se ha enviado un valor que no esta en catalogos. Contacte al
                                                                                                                administrador ";
147                             break;
148                       default:
149                                 this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                    Contacte al administrador ";
150                       } // end switch
151                       //connectionPool = null;
152                       this.estatus=false;
153                     } catch(SOAPException e) {
154                e.printStackTrace();
155           } // end catch
156                     }// end if
157                     return (this.estatus);
158           }
159
160
161           public SOAPMessage respuesta() {
162                   if (this.estatus)
163                   {
164                           return this.islaDatosXML;
165                   } else {
166                           return mensajeError();
167                   }// end if
168           }
169
170               public SOAPMessage mensajeError() {
171         SOAPMessage message = null;
172
173               try {
174                   // create price list message
175                   message = fac.createMessage();
176
177
178                   // Access the SOAPBody object
179                   SOAPPart part = message.getSOAPPart();
```

```
180                 SOAPEnvelope envelope = part.getEnvelope();
181                 SOAPBody body = envelope.getBody();
182                             SOAPHeader header = envelope.getHeader();
183                             header.detachNode();
184
185                             SOAPFactory soapFactory = SOAPFactory.newInstance();
186                             SOAPFault fault = body.addFault();
187                             Name faultName =
188                                     soapFactory.createName("Server",
189                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
190                             fault.setFaultCode(faultName);
191                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
192                             fault.setFaultString(this.codigoError);
193                 message.saveChanges();
194
195             } catch(Exception e) {
196                 e.printStackTrace();
197             }
198             return message;
199         }// end servicioError
200
201
202             public String getAgenciaId() {
203                     return this.agenciaid;
204             }
205
206             public String getPassword() {
207                     return this.password;
208             }
209
210             public boolean getEstatus() {
211                     return this.estatus;
212             }
213
214             public String getCodigoError(){
215                     return this.codigoError;
216             }
217
218             public void setCodigoError(String codigoError) {
219                     this.codigoError=codigoError;
220             }
221
222     }
```

ReportaVenta.java

```
1       // Bibliotecas para validar el mensaje XML SOAP
2       import org.iso_relax.verifier.*;
3       import com.sun.msv.driver.textui.ReportErrorHandler;
4       import org.xml.sax.*;
5
6       // Bibliotecas para procesar el mensaje XML SOAP
7       import java.io.*;
8       import javax.xml.soap.*;
9       import java.util.*;
10
11      // Bibliotecas para manejar la conexión de sql
12      import java.sql.*;
13      import coreservlets.*;
14
15      public class ReportaVenta
16      {
17              private String agenciaid="";
18              private String password="";
19              private String serieVehiculo="";
20              private String rfc_o_curp_cliente ="";
21              private String tipodeventa ="";
22              private String tipodepago ="";
23              private String rfcVendedor ="";
24              private String km ="";
25              private String fechaEntrega ="";
26              private String term ="";
27              private String tasa ="";
28              private String pagoMensual ="";
29              private String serieCambio ="";
```

```
30              private String kmCambio ="";
31              private boolean estatus = false;
32              private String codigoError = "5010-No hay datos para procesar";
33          static MessageFactory fac = null;
34
35          static {
36              try {
37                  fac = MessageFactory.newInstance();
38              } catch (Exception ex) {
39                  ex.printStackTrace();
40              }
41          }
42
43              public ReportaVenta(MimeHeaders headers, InputStream is)   throws IOException {
44                      try
45                      {
46                              SOAPMessage msg = fac.createMessage(headers, is);
47                      SOAPPart part = msg.getSOAPPart();
48
49                              // Código Agregado para la versión 2.0 del sistema
50                              // para validar datos desde la llegada del xml
51                              VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
52                                                                                                                      Schema schema =
                                        factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/ReportaVenta.xsd");
53                              Verifier verifier = schema.newVerifier();
54                              verifier.verify(part);
55                              // fin de código para version 2.0
56
57                              SOAPEnvelope envelope = part.getEnvelope();
58                              SOAPBody body = envelope.getBody();
59                              java.util.Iterator iterator = body.getChildElements();
60                              SOAPBodyElement addUnit = (SOAPBodyElement)iterator.next();
61                              iterator = addUnit.getChildElements();
62                              SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
63                              SOAPBodyElement password = (SOAPBodyElement)iterator.next();
64                              SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
65                              SOAPBodyElement rfc_o_curp_cliente = (SOAPBodyElement)iterator.next();
66                              SOAPBodyElement tipodeventa = (SOAPBodyElement)iterator.next();
67                              SOAPBodyElement tipodepago = (SOAPBodyElement)iterator.next();
68                              SOAPBodyElement rfcVendedor = (SOAPBodyElement)iterator.next();
69                              SOAPBodyElement km = (SOAPBodyElement)iterator.next();
70                              SOAPBodyElement fechaEntrega = (SOAPBodyElement)iterator.next();
71                              SOAPBodyElement term = (SOAPBodyElement)iterator.next();
72                              SOAPBodyElement tasa = (SOAPBodyElement)iterator.next();
73                              SOAPBodyElement pagoMensual = (SOAPBodyElement)iterator.next();
74                              SOAPBodyElement serieCambio = (SOAPBodyElement)iterator.next();
75                              SOAPBodyElement kmCambio = (SOAPBodyElement)iterator.next();
76                              this.agenciaid=agenciaid.getValue();
77                              this.password=password.getValue();
78                              this.serieVehiculo = serieVehiculo.getValue();
79                              this.rfc_o_curp_cliente = rfc_o_curp_cliente.getValue();
80                              this.tipodeventa=tipodeventa.getValue();
81                              this.tipodepago=tipodepago.getValue();
82                              this.rfcVendedor=rfcVendedor.getValue();
83                              this.km=km.getValue();
84                              this.fechaEntrega=fechaEntrega.getValue();
85                              this.term=term.getValue();
86                              this.tasa=tasa.getValue();
87                              this.pagoMensual=pagoMensual.getValue();
88                              this.serieCambio=serieCambio.getValue();
89                              this.kmCambio=kmCambio.getValue();
90                      }
91                      catch (SOAPException e)
92                      {
93                              this.estatus=false;
94                              this.codigoError="6000-Error al leer el xml de entrada en "+this.getClass().getName();
95                      }
96                      catch (VerifierConfigurationException e) // catch requerido para version 2.0
97                      {
98                              this.estatus=false;
99                              this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
100                     }
101                     catch (SAXException e) // catch requerido para version 2.0
102                     {
103                             this.estatus=false;
104                             this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
105                     }
106             }
107
```

```
108         public ReportaVenta(String agenciaid, String password,
109                 String serieVehiculo, String rfc_o_curp_cliente,
110                 String tipodeventa, String tipodepago) {
111                 this.agenciaid=agenciaid;
112                 this.password=password;
113                 this.serieVehiculo=serieVehiculo;
114                 this.rfc_o_curp_cliente = rfc_o_curp_cliente;
115                 this.tipodeventa=tipodeventa;
116                 this.tipodepago=tipodepago;
117         }
118
119         public boolean ejecuta(ConnectionPool connectionPool) {
120                 String strSQL;
121                 // if requerido para version 2.0
122                 if (this.codigoError.equals( "5010-No hay datos para procesar"))
123                 {
124                 try {
125
126                         Connection miconexion = connectionPool.getConnection();
127                         String comilla ="'";
128                         ResultSet rs ,rs2;
129                         String serieVehiculo;
130                         PreparedStatement pstm;
131                         Statement stm = miconexion.createStatement();
132                         strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
133                                  "WHERE agenciaid = " + comilla + agenciaid + comilla +
134                                       "AND password = password("+comilla + password + comilla +")";
135                         rs = stm.executeQuery(strSQL);
136                         rs.next();
137                         if (rs.getInt("conteo")!=1) {// if #1
138                                 this.estatus=false;
139                             this.codigoError="5000 - agenciaid:" + agenciaid +
140                                         " con password:" + password + " no son validos. Contacte al administrador ";
141                         } else {
142                                 strSQL = "SELECT count(*) as conteo FROM tblVendedores "+
143                                                  " WHERE agenciaid ="+comilla + this.agenciaid+ comilla +
144                                                  " AND rfcVendedor=" + comilla + this.rfcVendedor + comilla ;
145                                 rs = stm.executeQuery(strSQL);
146                                 rs.next();
147                                 if (rs.getInt("conteo")!=1) { // if #2
148                                         this.estatus=false;
149                                     this.codigoError="5120 - El vendedor " + this.rfcVendedor +
150                                         " no ha sido dado de alta para la agencia " + this.agenciaid +
151                                         ". Utilize el metodo AltaVendedor ";
152                                 } else {
153                                         strSQL = "SELECT count(*) as conteo FROM tblClientes " +
154                                                          " WHERE agenciaid ="+comilla + this.agenciaid+ comilla +
155                                                                  " AND rfc_o_curp_cliente=" + comilla +
                                                                     this.rfc_o_curp_cliente + comilla ;
156                                         rs = stm.executeQuery(strSQL);
157                                         rs.next();
158                                         if (rs.getInt("conteo")!=1) {// if #3
159                                                 this.estatus=false;
160                                             this.codigoError="5130 - El cliente " + this.rfc_o_curp_cliente +
161                                                 " no ha sido dado de alta para la agencia " + this.agenciaid +
162                                                 ". Utilice el metodo AltaCliente ";
163                                         } else {
164                                                 strSQL = "SELECT count(*) as conteo FROM tblInventario " +
165                                                                  " WHERE serieVehiculo=" + comilla +
                                                                     this.serieVehiculo + comilla ;
166                                                 rs = stm.executeQuery(strSQL);
167                                                 rs.next();
168                                                 if (rs.getInt("conteo")!=1) {// if #4
169                                                         this.estatus=false;
170                                                           this.codigoError="5140 - El vehiculo " + this.serieVehiculo +
171                                                         " no ha sido dado de alta en el inventario. " +
172                                                         "Utilice el metodo AltaUnidad o AltaInventario ";
173                                                 } else {
174                                                         strSQL = "SELECT agenciaid FROM tblInventario "+
175                                                                  " WHERE serieVehiculo=" + comilla +
                                                                     this.serieVehiculo + comilla ;
176                                                         rs = stm.executeQuery(strSQL);
177                                                         rs.next();
178                                                         String agenciaPertenece = rs.getString("agenciaid");
179                                                         if (!agenciaPertenece.equals(this.agenciaid)) {// if #5
180                                                                 this.estatus=false;
181                                                                     this.codigoError="5150 - El vehiculo " +
                                                                         this.serieVehiculo +
```

```
182                                                 " pertenece a la agencia " + agenciaPertenece +
183                                                 " y no se puede reportar la venta. " +
184                                                    "Utilize el metodo Transferencia para ingresar la
                                                                                    unidad a la agencia " +
185                                                 this.agenciaid;
186                                             } else {
187                                                             strSQL = "SELECT count(*) as conteo FROM
                                                                                                tbltipodeventa "+
188                                                   " WHERE tipodeventa=" + comilla + this.tipodeventa
                                                                                                    + comilla ;
189                                                 rs = stm.executeQuery(strSQL);
190                                                 rs.next();
191                                                 if (rs.getInt("conteo")!=1) {// if #6
192                                                         this.estatus=false;
193                                                          this.codigoError="5160 - El tipo de venta "
                                                                                        + this.tipodeventa +
194                                                         " es invalido. " ;
195                                                 } else {
196                                                     strSQL = "SELECT count(*) as conteo FROM
                                                                                            tbltipodepago "+
197                                                             " WHERE tipodepago=" + comilla +
                                                                    this.tipodepago + comilla ;
198                                                 rs = stm.executeQuery(strSQL);
199                                                 rs.next();
200                                                 if (rs.getInt("conteo")!=1) {// if #7
201                                                         this.estatus=false;
202                                                         this.codigoError="5170 - El tipo de
                                                                  pago " + this.tipodepago +
203                                                         " es invalido. " ;
204                                                 } else {
205                                                         this.estatus=true;
206                                                         strSQL = "SELECT count(*) as conteo
                                                                                        FROM tblVentas "+
207                                                            " WHERE serieVehiculo=" + comilla
                                                             + this.serieVehiculo + comilla ;
208                                                         rs = stm.executeQuery(strSQL);
209                                                         rs.next();
210                                                          if (rs.getInt("conteo")==1) {// if
                                                                                                #8
211                                                                 strSQL = "SELECT agenciaid
                                                                        FROM tblVentas "+
212                                                                  " WHERE serieVehiculo=" +
                                                             comilla + this.serieVehiculo + comilla ;
213                                                                                             rs =
                                                                   stm.executeQuery(strSQL);
214                                                                rs.next();
215                                                                this.estatus=false;
216                                                                   this.codigoError="5170 -
                                                                    La venta del vehiculo " +
217                                                                        this.serieVehiculo
                                                        + " ha sido previamente reportada por " +
218
                                        rs.getString("agenciaid") + ". Contacte con esta agencia para mas información";
219                                                         } else {
220                                                             strSQL = "INSERT INTO tblVentas
                                                    (serieVehiculo, rfc_o_curp_cliente, agenciaid,"+
221                                                                                    "tipodeventa,
                                                tipodepago, rfcvendedor, km, fechaEntrega, term, tasa,"+
222
                                                        "pagoMensual,serieCambio, kmCambio) VALUES "+
223                                                                                    "("+comilla +
                                                                this.serieVehiculo + comilla +
224                                                                                    "," + comilla +
                                                        this.rfc_o_curp_cliente + comilla +
225                                                                                    "," + comilla +
                                                                  this.agenciaid + comilla +
226                                                                                    "," + comilla +
                                                                this.tipodeventa + comilla +
227                                                                                    "," + comilla +
                                                                 this.tipodepago + comilla +
228                                                                                    "," + comilla +
                                                                this.rfcVendedor + comilla +
229                                                                                    "," + this.km +
230                                                                                    "," + comilla +
                                                              this.fechaEntrega + comilla +
231                                                                                    "," + this.term +
232                                                                                    "," + this.tasa +
233                                                                                           "," +
                                                                                    this.pagoMensual +
234                                                                                    "," + comilla +
```

```
                                                                                                                    this.serieCambio + comilla +
235                                                                                                                                          "," +
                                                                                                                                this.kmCambio + ")";
236                                                                                                                                          pstm =
                                                                                                                              miconexion.prepareStatement(strSQL);
237                                                                                                                               pstm.execute();
238                                                                                                                               } //end if #8
239                                                                                                                         }// end if #7
240                                                                                                                   }// end if #6
241                                                                                                             }// end if #5
242                                                                                                       }// end if #4
243                                                                                                 }// end if #3
244                                                                                           }// end if #2
245                                                                                     }//end if #1
246                                                                                     rs.close();
247                                                                                     miconexion.close();
248                                                                               } catch(SQLException sqle) {
249                                                                                 switch (sqle.getErrorCode())
250                                                                                 {
251                                                                                 case 0:
252                                                                                   this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
253                                                                                        break;
254                                                                                 case 1049:
255                                                                                   this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
256                                                                                        break;
257                                                                                 case 1216: //para fks multiples se tendra que revisar que datos no son validos
258                                                                                     this.codigoError="1216 - La agencia " + this.agenciaid + " no existe en el catalogo. Contacte al
                                                                                                                                                                    administrador ";
259                                                                                        break;
260                                                                                 default:
261                                                                                          this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                                                                                                                  Contacte al administrador ";
262                                                                                 }//end switch
263                                                                                 //connectionPool = null;
264                                                                                 this.estatus=false;
265                                                                               }//end catch
266                                                                               }//end if
267                                                                               return (this.estatus);
268                                                                       }
269
270
271                                                                       public SOAPMessage respuesta() {
272                                                                               if (this.estatus)
273                                                                               {
274                                                                                       return mensajeExitoso();
275                                                                               } else {
276                                                                                       return mensajeError();
277                                                                               }// end if
278                                                                       }
279
280                                                                       public SOAPMessage mensajeExitoso() {
281                                                                   SOAPMessage message = null;
282
283                                                                       try {
284                                                                           // create price list message
285                                                                           message = fac.createMessage();
286
287                                                                           // Access the SOAPBody object
288                                                                           SOAPPart part = message.getSOAPPart();
289                                                                           SOAPEnvelope envelope = part.getEnvelope();
290                                                                           SOAPBody body = envelope.getBody();
291                                                                                       SOAPHeader header = envelope.getHeader();
292                                                                                       header.detachNode();
293
294                                                                           Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
295                                                                               "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
296                                                                           SOAPBodyElement list = body.addBodyElement(bodyName);
297
298                                                                           Name exitosoN = envelope.createName("Respuesta");
299                                                                           SOAPElement exitoso = list.addChildElement(exitosoN);
300                                                                                       exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
301                                                                           message.saveChanges();
302
303                                                                       } catch(Exception e) {
304                                                                           e.printStackTrace();
305                                                                       }
306                                                                       return message;
307                                                                   }// end mensajeExitoso
308
```

```
309             public SOAPMessage mensajeError() {
310         SOAPMessage message = null;
311
312             try {
313                 // create price list message
314                 message = fac.createMessage();
315
316                 // Access the SOAPBody object
317                 SOAPPart part = message.getSOAPPart();
318                 SOAPEnvelope envelope = part.getEnvelope();
319                 SOAPBody body = envelope.getBody();
320                             SOAPHeader header = envelope.getHeader();
321                             header.detachNode();
322
323                             SOAPFactory soapFactory = SOAPFactory.newInstance();
324                             SOAPFault fault = body.addFault();
325                             Name faultName =
326                                     soapFactory.createName("Server",
327                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
328                             fault.setFaultCode(faultName);
329                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                    clase
330                             fault.setFaultString(this.codigoError);
331                 message.saveChanges();
332
333             } catch(Exception e) {
334                 e.printStackTrace();
335             }
336             return message;
337         }// end servicioError
338
339
340             public String getAgenciaId() {
341                     return this.agenciaid;
342             }
343
344             public String getPassword() {
345                     return this.password;
346             }
347
348             public boolean getEstatus() {
349                     return this.estatus;
350             }
351
352             public String getCodigoError(){
353                     return this.codigoError;
354             }
355
356             public void setCodigoError(String codigoError) {
357                     this.codigoError=codigoError;
358             }
359
360     }
```

ServicioHTTP.java

```
1       import java.io.*;
2       import javax.servlet.*;
3       import javax.servlet.http.*;
4       import javax.xml.soap.*;
5       import java.util.*;
6       import java.sql.*;
7       import coreservlets.*;
8
9       /* Base de Datos MySQL con Interfaz Java Web Services.
10         Caso de Estudio: Recepcion de Ordenes de Reparacion y Reportes de Ventas de La Industria Automotriz
11         Version: 1.0 SERVER apr-13-2004
12          Procesamiento de XML SOAP revisando integridad referencial de la Base de datos
13         Version: 2.0 SERVER jul-01-2004
14          XML Schemas para validacion de estructuras y datos de XML SOAP
15         Tesis DEPFI-UNAM SECCION ELECTRICA
16         Direccion:   ANA MARIA VAZQUEZ VARGAS VAVA480918-165
17         Elaboracion: MORENO MOTTE SAUL 09201836-9
18      */
19      public class ServicioHTTP extends HttpServlet {
20              FileOutputStream fos;
21              OutputStreamWriter osw;
22              public ConnectionPool connectionPool;
```

```
23
24            // metodo init de los servlets
25            // se utiliza para dar de alta al "pool" de conexiones para la base de datos
26            public void init() {
27                    String driver = "com.mysql.jdbc.Driver";
28                      String connString      =
29                      "jdbc:mysql://localhost/produccion?user=root&password=";
30                    try {
31                      connectionPool =
32                            new ConnectionPool(driver, connString,
33                                                                    initialConnections(),
34                                                                    maxConnections(),
35                                                                    true);
36                    } catch(SQLException sqle) {
37                      System.err.println("Error making pool: " + sqle);
38                      getServletContext().log("Error making pool: " + sqle);
39                      connectionPool = null;
40                    }
41            } //end init
42
43
44            protected int initialConnections() {
45            return(10);
46            }
47
48
49            protected int maxConnections() {
50            return(50);
51            }
52
53
54
55
56                public void doGet(HttpServletRequest request, HttpServletResponse response)
57          throws IOException, ServletException
58          {
59                        response.setHeader("Content-Type", "text/xml; charset=utf-8");
60                        response.setHeader("Content-Length", "400");
61                        javax.servlet.ServletOutputStream sos = response.getOutputStream();
62                        try
63                        {
64                                SOAPMessage reply = null;
65                                AltaUnidad objeto1 = new AltaUnidad("Saul","Moreno","Motte");
66                                   objeto1.setCodigoError("5020 - Se debe utilizar ServicioHTTP con APIs de SOAP (XML POST via
                                                                                                                        HTTP)");
67                                reply = objeto1.respuesta();
68                                reply.writeTo(sos);
69                        }
70                        catch (Exception ex) {
71                   throw new ServletException("SAAJ POST failed: " +  ex.getMessage());
72                }
73                        sos.close();
74
75                }
76
77          public void doPost(HttpServletRequest request, HttpServletResponse response)
78          throws IOException, ServletException
79          {
80
81                        //=============================================================================================
82                        // leer el nombre del metodo y los parametros del SOAPRequest
83                        //=============================================================================================
84
85                        // Obtener solamente el nombre del metodo RPC de los headers
86                        String soapActionString = request.getHeader("soapaction");
87                        soapActionString=soapActionString.substring(1,soapActionString.length()-1);
88                        //osw.write("soapaction\n"+soapActionString+"\n");
89
90                        // Get all the headers from the HTTP request
91                    MimeHeaders headers = getHeaders(request);
92
93                        // Get the body of the HTTP request
94                        InputStream is = request.getInputStream();
95
96                        // Preparar la respuesta para todos lo metodos
97                        response.setHeader("Content-Type", "text/xml; charset=utf-8");
98                        javax.servlet.ServletOutputStream sos = response.getOutputStream();
99                        SOAPMessage reply = null;
100
```

```
101                     try
102                     {
103                             if (soapActionString.equals("http://www.unam.mx/AltaUnidad")){
104                                     AltaUnidad objeto = new AltaUnidad(headers,is);
105                                     objeto.ejecuta(connectionPool);
106                                     reply = objeto.respuesta();
107                             } else if (soapActionString.equals("http://www.unam.mx/BajaUnidad")) {
108                                     BajaUnidad objeto = new BajaUnidad(headers,is);
109                                     objeto.ejecuta(connectionPool);
110                                     reply = objeto.respuesta();
111                             } else if (soapActionString.equals("http://www.unam.mx/AltaInventario")) {
112                                     AltaInventario objeto = new AltaInventario(headers,is);
113                                     objeto.ejecuta(connectionPool);
114                                     reply = objeto.respuesta();
115                             } else if (soapActionString.equals("http://www.unam.mx/AltaVendedor")) {
116                                     AltaVendedor objeto = new AltaVendedor(headers,is);
117                                     objeto.ejecuta(connectionPool);
118                                     reply = objeto.respuesta();
119                             } else if (soapActionString.equals("http://www.unam.mx/BajaVendedor")) {
120                                     BajaVendedor objeto = new BajaVendedor(headers,is);
121                                     objeto.ejecuta(connectionPool);
122                                     reply = objeto.respuesta();
123                             } else if (soapActionString.equals("http://www.unam.mx/ListaVendedores")) {
124                                     ListaVendedores objeto = new ListaVendedores(headers,is);
125                                     objeto.ejecuta(connectionPool);
126                                     reply = objeto.respuesta();
127                             } else if (soapActionString.equals("http://www.unam.mx/AltaCliente")) {
128                                     AltaCliente objeto = new AltaCliente(headers,is);
129                                     objeto.ejecuta(connectionPool);
130                                     reply = objeto.respuesta();
131                             } else if (soapActionString.equals("http://www.unam.mx/BajaCliente")) {
132                                     BajaCliente objeto = new BajaCliente(headers,is);
133                                     objeto.ejecuta(connectionPool);
134                                     reply = objeto.respuesta();
135                             } else if (soapActionString.equals("http://www.unam.mx/ListaClientes")) {
136                                     ListaClientes objeto = new ListaClientes(headers,is);
137                                     objeto.ejecuta(connectionPool);
138                                     reply = objeto.respuesta();
139                             } else if (soapActionString.equals("http://www.unam.mx/ReportaVenta")) {
140                                     ReportaVenta objeto = new ReportaVenta(headers,is);
141                                     objeto.ejecuta(connectionPool);
142                                     reply = objeto.respuesta();
143                             } else if (soapActionString.equals("http://www.unam.mx/CancelaVenta")) {
144                                     CancelaVenta objeto = new CancelaVenta(headers,is);
145                                     objeto.ejecuta(connectionPool);
146                                     reply = objeto.respuesta();
147                             } else if (soapActionString.equals("http://www.unam.mx/Transferencia")) {
148                                     Transferencia objeto = new Transferencia(headers,is);
149                                     objeto.ejecuta(connectionPool);
150                                     reply = objeto.respuesta();
151                             } else if (soapActionString.equals("http://www.unam.mx/AltaAsesor")) {
152                                     AltaAsesor objeto = new AltaAsesor(headers,is);
153                                     objeto.ejecuta(connectionPool);
154                                     reply = objeto.respuesta();
155                             } else if (soapActionString.equals("http://www.unam.mx/BajaAsesor")) {
156                                     BajaAsesor objeto = new BajaAsesor(headers,is);
157                                     objeto.ejecuta(connectionPool);
158                                     reply = objeto.respuesta();
159                             } else if (soapActionString.equals("http://www.unam.mx/ListaAsesores")) {
160                                     ListaAsesores objeto = new ListaAsesores(headers,is);
161                                     objeto.ejecuta(connectionPool);
162                                     reply = objeto.respuesta();
163                             } else if (soapActionString.equals("http://www.unam.mx/AltaOrden")) {
164                                     AltaOrden objeto = new AltaOrden(headers,is);
165                                     objeto.ejecuta(connectionPool);
166                                     reply = objeto.respuesta();
167                             } else if (soapActionString.equals("http://www.unam.mx/ImportesOrden")) {
168                                     ImportesOrden objeto = new ImportesOrden(headers,is);
169                                     objeto.ejecuta(connectionPool);
170                                     reply = objeto.respuesta();
171                             } else if (soapActionString.equals("http://www.unam.mx/CierraOrden")) {
172                                     CierraOrden objeto = new CierraOrden(headers,is);
173                                     objeto.ejecuta(connectionPool);
174                                     reply = objeto.respuesta();
175                             } else if (soapActionString.equals("http://www.unam.mx/CancelaOrden")) {
176                                     CancelaOrden objeto = new CancelaOrden(headers,is);
177                                     objeto.ejecuta(connectionPool);
178                                     reply = objeto.respuesta();
179                             }//end if
```

```
180
181                         reply.writeTo(sos);
182                         sos.close();
183                 }
184                 catch (javax.xml.soap.SOAPException ex) {
185                         //fos = new FileOutputStream("ServicioHTTP.LOG");
186                         //osw = new OutputStreamWriter(fos);
187                         //osw.write(ex.getMessage());
188                         //osw.close();
189                         //fos.close();
190                         throw new ServletException("SAAJ POST failed: " +  ex.getMessage());
191                 } // end catch
192
193
194
195
196
197
198
199     }//end doPost
200
201     static MimeHeaders getHeaders(HttpServletRequest req) {
202
203         Enumeration enum = req.getHeaderNames();
204         MimeHeaders headers = new MimeHeaders();
205
206         while (enum.hasMoreElements()) {
207             String headerName = (String)enum.nextElement();
208             String headerValue = req.getHeader(headerName);
209
210             StringTokenizer values =
211                 new StringTokenizer(headerValue, ",");
212             while (values.hasMoreTokens()) {
213                 headers.addHeader(headerName,
214                     values.nextToken().trim());
215             }
216         }
217         return headers;
218     }//end getHeaders
219
220     static void putHeaders(MimeHeaders headers,
221             HttpServletResponse res) {
222
223         Iterator it = headers.getAllHeaders();
224         while (it.hasNext()) {
225             MimeHeader header = (MimeHeader)it.next();
226
227             String[] values = headers.getHeader(header.getName());
228             if (values.length == 1) {
229                 res.setHeader(header.getName(), header.getValue());
230             } else {
231                 StringBuffer concat = new StringBuffer();
232                 int i = 0;
233                 while (i < values.length) {
234                     if (i != 0) {
235                         concat.append(',');
236                     }
237                     concat.append(values[i++]);
238                 }
239                 res.setHeader(header.getName(), concat.toString());
240             }
241         }
242     }//end putHeaders
243
244          static void escribeHeaders(MimeHeaders headers, OutputStreamWriter osw)
245                  throws java.io.IOException
246                  {
247                  //esta rutina es un clon de putHeaders, la utilize solo para ver que headers trae
248                  //el soap request. El header que me interesa es el soapaction porque me dice que
249                  //metodo web tiene que ejecutar el servicio.
250         Iterator it = headers.getAllHeaders();
251         while (it.hasNext()) {
252             MimeHeader header = (MimeHeader)it.next();
253
254             String[] values = headers.getHeader(header.getName());
255             if (values.length == 1) {
256                 osw.write(header.getName()+"\n");
257                                 osw.write(header.getValue()+"\n");
258                                 osw.write("---"+"\n");
```

```
259                 } else {
260                     StringBuffer concat = new StringBuffer();
261                     int i = 0;
262                     while (i < values.length) {
263                         if (i != 0) {
264                             concat.append(',');
265                         }
266                         concat.append(values[i++]);
267                     }
268                     osw.write(header.getName()+"\n");
269                                     osw.write(concat.toString()+"\n");
270                                     osw.write("---"+"\n");
271                 }
272                      }//end while
273             }//end escribeHeaders
274     }
```

Telefono.java

```
1     public class Telefono
2     {
3             private String tipoDeTelefono="";
4             private String lada="";
5             private String numero="";
6
7             public Telefono(String tipoDeTelefono, String lada, String numero) {
8                     this.tipoDeTelefono=tipoDeTelefono;
9                     this.lada=lada;
10                    this.numero=numero;
11            }
12
13
14            public String getTipoDeTelefono(){
15                    return this.tipoDeTelefono;
16            }
17
18            public void setTipoDeTelefono(String tipoDeTelefono) {
19                    this.tipoDeTelefono=tipoDeTelefono;
20            }
21
22            public String getLada(){
23                    return this.lada;
24            }
25
26            public void setLada(String lada) {
27                    this.lada=lada;
28            }
29
30            public String getNumero(){
31                    return this.numero;
32            }
33
34            public void setNumero(String numero) {
35                    this.numero=numero;
36            }
37
38    }
```

Transferencia.java

```
1     // Bibliotecas para validar el mensaje XML SOAP
2     import org.iso_relax.verifier.*;
3     import com.sun.msv.driver.textui.ReportErrorHandler;
4     import org.xml.sax.*;
5
6     // Bibliotecas para procesar el mensaje XML SOAP
7     import java.io.*;
8     import javax.xml.soap.*;
9     import java.util.*;
10
11    // Bibliotecas para manejar la conexión de sql
12    import java.sql.*;
13    import coreservlets.*;
14
15    public class Transferencia
```

```
16  {
17          private String agenciaid="";
18          private String password="";
19          private String serieVehiculo="";
20          private String agenciaidorigen="";
21          private String agenciaiddestino="";
22          private boolean estatus = false;
23          private String codigoError = "5010-No hay datos para procesar";
24      static MessageFactory fac = null;
25
26      static {
27          try {
28              fac = MessageFactory.newInstance();
29          } catch (Exception ex) {
30              ex.printStackTrace();
31          }
32      }
33
34          public Transferencia(MimeHeaders headers, InputStream is)  throws IOException {
35                  try
36                  {
37                          SOAPMessage msg = fac.createMessage(headers, is);
38              SOAPPart part = msg.getSOAPPart();
39
40                          // Código Agregado para la versión 2.0 del sistema
41                          // para validar datos desde la llegada del xml
42                          VerifierFactory factory = new com.sun.msv.verifier.jarv.TheFactoryImpl();
43                                                                                                  Schema schema =
                                    factory.compileSchema("http://localhost:8080/produccion/XSDSchemas/Transferencia.xsd");
44                          Verifier verifier = schema.newVerifier();
45                          verifier.verify(part);
46                          // fin de código para version 2.0
47
48                          SOAPEnvelope envelope = part.getEnvelope();
49              SOAPBody body = envelope.getBody();
50                          java.util.Iterator iterator = body.getChildElements();
51                          SOAPBodyElement transferencia = (SOAPBodyElement)iterator.next();
52                          iterator = transferencia.getChildElements();
53                          SOAPBodyElement agenciaid = (SOAPBodyElement)iterator.next();
54                          SOAPBodyElement password = (SOAPBodyElement)iterator.next();
55                          SOAPBodyElement serieVehiculo = (SOAPBodyElement)iterator.next();
56                          SOAPBodyElement agenciaiddestino = (SOAPBodyElement)iterator.next();
57                          this.agenciaid=agenciaid.getValue();
58                          this.password=password.getValue();
59                          this.serieVehiculo = serieVehiculo.getValue();
60                          this.agenciaiddestino = agenciaiddestino.getValue();
61                  }
62                  catch (SOAPException e)
63                  {
64                          this.estatus=false;
65                          this.codigoError="6000-Error al leer el xml de entrada en AltaUnidad";
66                  }
67                  catch (VerifierConfigurationException e) // catch requerido para version 2.0
68                  {
69                          this.estatus=false;
70                          this.codigoError="7000-Error en las configuración de los XSD Schemas. ";
71                  }
72                  catch (SAXException e) // catch requerido para version 2.0
73                  {
74                          this.estatus=false;
75                          this.codigoError="7001-El XML es inválido. Detalle:"+e.getMessage();
76                  }
77
78          }
79
80          public Transferencia(String agenciaid, String password,
81                  String serieVehiculo, String agenciaiddestino) {
82                  this.agenciaid=agenciaid;
83                  this.password=password;
84                  this.serieVehiculo=serieVehiculo;
85                  this.agenciaiddestino = agenciaiddestino;
86          }
87
88          public boolean ejecuta(ConnectionPool connectionPool) {
89                  String strSQL;
90                  // if requerido para version 2.0
91                  if (this.codigoError.equals( "5010-No hay datos para procesar"))
92                  {
93                  try {
```

```
94
95                          Connection miconexion = connectionPool.getConnection();
96                          miconexion.setAutoCommit(false);
97                          PreparedStatement pstm;
98                          ResultSet rs;
99                          String comilla ="'";
100                         Statement stm = miconexion.createStatement();
101                         strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
102                                  " WHERE agenciaid = " + comilla + agenciaid + comilla +
103                                       " AND password = password("+comilla + password + comilla+")" ;
104                         rs = stm.executeQuery(strSQL);
105                         rs.next();
106                         if (rs.getInt("conteo")!=1) // if #1
107                         {
108                                 this.estatus=false;
109                             this.codigoError="5000 - agenciaid " + agenciaid +
110                                         " con password " + password + " no son validos. Contacte al administrador ";
111                         } else {
112                                 strSQL = "SELECT count(*) as conteo FROM tblInventario " +
113                                         " WHERE serieVehiculo = " +       comilla + this.serieVehiculo + comilla ;
114                                 rs = stm.executeQuery(strSQL);
115                                 rs.next();
116                                 if (rs.getInt("conteo")==0) // if #2
117                                 {
118                                         this.estatus=false;
119                                     this.codigoError="5500 - Error: La unidad" + this.serieVehiculo +
120                                         " no existe en el inventario. Utilize la transaccion AltaUnidad";
121                                 } else {
122                                         strSQL = "SELECT count(*) as conteo FROM tblVentas " +
123                                                  " WHERE serieVehiculo = " +       comilla + this.serieVehiculo +
                                                                                                                  comilla ;
124                                         rs = stm.executeQuery(strSQL);
125                                         rs.next();
126                                         if (rs.getInt("conteo")==1) // if #3
127                                         {
128                                                 this.estatus=false;
129                                                 strSQL = "SELECT agenciaid FROM tblVentas " +
130                                                                          " WHERE serieVehiculo = " +       comilla +
                                                                                  this.serieVehiculo + comilla ;
131                                                 rs = stm.executeQuery(strSQL);
132                                                 rs.next();
133                                                 this.codigoError="5510 - Error: La unidad" + this.serieVehiculo +
134                                                                  " ha sido reportada como vendida por la agencia " +
                                                                                      rs.getString("agenciaid")+
135                                                    " (sugerencia: hable a esa agencia para que envien una transaccion
                                                                                    CancelaVenta)";
136                                         } else {
137                                                 strSQL = "SELECT count(*) as conteo FROM tblAgencias " +
138                                                                                  " WHERE agenciaid = " + comilla +
                                                                                  this.agenciaiddestino + comilla ;
139                                                 rs = stm.executeQuery(strSQL);
140                                                 rs.next();
141                                                 if (rs.getInt("conteo")!=1) // if #4
142                                                 {
143                                                         this.estatus=false;
144                                                                 this.codigoError="1217 - agenciaiddestino " +
                                                     this.agenciaiddestino + " no es valida. Contacte al administrador ";
145                                                 } else {
146
147                                                         strSQL = "SELECT agenciaid FROM tblInventario " +
148                                                                          " WHERE serieVehiculo = " +       comilla +
                                                                                  this.serieVehiculo + comilla ;
149                                                         rs = stm.executeQuery(strSQL);
150                                                         rs.next();
151                                                         this.agenciaidorigen=rs.getString("agenciaid");
152
153                                                         strSQL = "UPDATE tblInventario " +
154                                                                                  " SET agenciaid = " + comilla +
                                                                                  this.agenciaiddestino + comilla +
155                                                                  " WHERE serieVehiculo = " +       comilla +
                                                                                  this.serieVehiculo + comilla ;
156                                                         pstm = miconexion.prepareStatement(strSQL);
157                                                         pstm.execute();
158
159                                                                                  strSQL = "INSERT INTO
                                                                 tblTransferencias(serieVehiculo,agenciaid," +
160
                                                      "agenciaidorigen,agenciaiddestino,fechahora) VALUES " +
161                                                                  "(" + comilla + this.serieVehiculo +comilla
```

```
                                                                              +
                                                        "," + comilla + this.agenciaid + comilla +
                                                              "," + comilla + this.agenciaidorigen +
                                                                                              comilla +
                                                             "," + comilla + this.agenciaiddestino +
                                                                                              comilla +
                                                        "," + "now()"+")";
                                          pstm = miconexion.prepareStatement(strSQL);
                                          pstm.execute();
                                          pstm.close();
                                          this.estatus=true;
                                  }// end if #4
                          } // end if #3
                  }// end if #2
          }// end if #1
          rs.close();
          miconexion.commit(); // si la transaccion no genero error se guardaran los datos
          miconexion.close();
      } catch(SQLException sqle) {
        switch (sqle.getErrorCode())
        {
        case 0:
          this.codigoError="0000 - Motor de Base de Datos no disponible. Contacte al administrador ";
              break;
        case 1049:
          this.codigoError="1049 - Base de Datos no disponible. Contacte al administrador ";
              break;
        case 1062:
          this.codigoError="1062 - Unidad " + this.serieVehiculo + " ya se encuentra en Inventario ";
              break;
        case 1216: //para fks multiples se tendra que revisar que datos no son validos
                  this.codigoError="1216 - La agencia " + this.agenciaiddestino + " no existe en el catalogo.
                                                                           Contacte al administrador ";
              break;
        default:
                 this.codigoError=Integer.toString(sqle.getErrorCode()) + " - Error critico en base de datos.
                                                          Contacte al administrador ";
        }//end switch

        //connectionPool = null;
        this.estatus=false;
      }//end catch
      }//end if
      return (this.estatus);
  }


  public SOAPMessage respuesta() {
          if (this.estatus)
          {
                  return mensajeExitoso();
          } else {
                  return mensajeError();
          }// end if
  }

  public SOAPMessage mensajeExitoso() {
SOAPMessage message = null;

    try {
        // create price list message
        message = fac.createMessage();

        // Access the SOAPBody object
        SOAPPart part = message.getSOAPPart();
        SOAPEnvelope envelope = part.getEnvelope();
        SOAPBody body = envelope.getBody();
                  SOAPHeader header = envelope.getHeader();
                  header.detachNode();

        Name bodyName = envelope.createName(this.getClass().getName()+"Respuesta",
            "m", "http://www.unam.mx/"+this.getClass().getName()+"Respuesta");
        SOAPBodyElement list = body.addBodyElement(bodyName);

        Name exitosoN = envelope.createName("Respuesta");
        SOAPElement exitoso = list.addChildElement(exitosoN);
                  exitoso.addTextNode("9999-"+this.getClass().getName()+" es Exitoso");
        message.saveChanges();

```

```
236             } catch(Exception e) {
237                 e.printStackTrace();
238             }
239             return message;
240         }// end mensajeExitoso
241 
242             public SOAPMessage mensajeError() {
243         SOAPMessage message = null;
244 
245             try {
246                 // create price list message
247                 message = fac.createMessage();
248 
249 
250                 // Access the SOAPBody object
251                 SOAPPart part = message.getSOAPPart();
252                 SOAPEnvelope envelope = part.getEnvelope();
253                 SOAPBody body = envelope.getBody();
254                             SOAPHeader header = envelope.getHeader();
255                             header.detachNode();
256 
257                             SOAPFactory soapFactory = SOAPFactory.newInstance();
258                             SOAPFault fault = body.addFault();
259                             Name faultName =
260                                     soapFactory.createName("Server",
261                                     "",SOAPConstants.URI_NS_SOAP_ENVELOPE);
262                             fault.setFaultCode(faultName);
263                              fault.setFaultActor("http://www.unam.mx/"+this.getClass().getName()); //el mismo nombre de la
                                                                                                                                clase
264                             fault.setFaultString(this.codigoError);
265                 message.saveChanges();
266 
267             } catch(Exception e) {
268                 e.printStackTrace();
269             }
270             return message;
271         }// end servicioError
272 
273 
274             public String getAgenciaId() {
275                     return this.agenciaid;
276             }
277 
278             public String getPassword() {
279                     return this.password;
280             }
281 
282             public String getSerieVehiculo() {
283                     return this.serieVehiculo;
284             }
285 
286             public boolean getEstatus() {
287                     return this.estatus;
288             }
289 
290             public String getCodigoError(){
291                     return this.codigoError;
292             }
293 
294             public void setCodigoError(String codigoError) {
295                     this.codigoError=codigoError;
296             }
297 
298     }
```

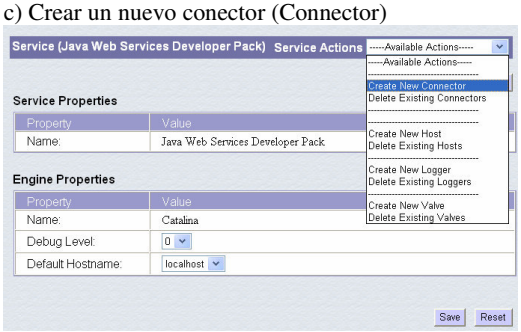
Anexo 6. Guía de Instalación de la Interfaz Java Web Services

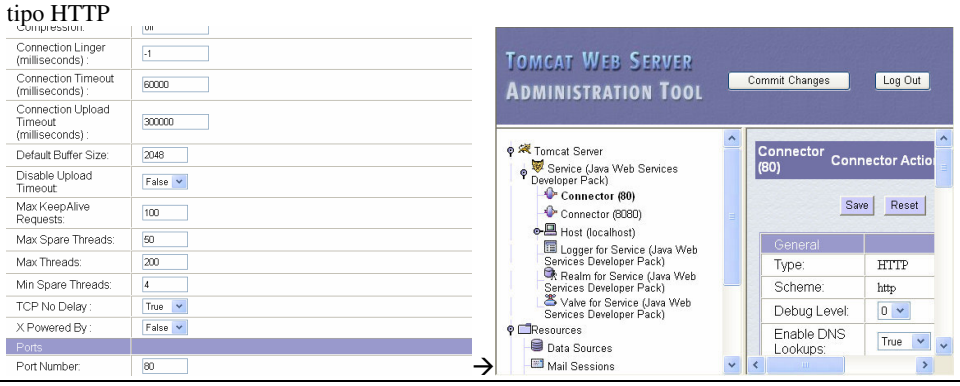
Instalación de Aplicación "Base de datos Mysql con Interfase Java Web Services. Caso de estudio: Recepción de reportes de ventas y órdenes de reparación de la industria automotriz"

1	Instalar Java 2 Platform Standard Edition (5 min)	Ejecutar **D:\Application_Toolkit\j2Sdk\j2sdk-1_4_2-windows-i586.exe**
2	Instalar Java 2 Documentation (3 min)	Descompactar **D:\Application_Toolkit\j2Sdk\j2sdk-1_4_2-doc.zip** En **C:\jwsdk1.4.2**
3	Acceso directo a Java 2 Documentation	Poner acceso directo en escritorio **C:\jwsdk1.4.2\docs\index.html**
4	Instalar Java Web Services Developers Package (5 min)	Ejecutar **D:\Application_Toolkit\web services\jwsdp-1_3-windows-i586.exe** Escribir User: **admin** Password : **admin** Password Confirmation: **admin.**
5	Instalar Java Web Services Documentation	Descompactar **D:\Application_Toolkit\web services\jwsdp-1_3_01-tutorial.zip** En **C:\jwsdp-1.3** Poner acceso directo en escritorio de **C:\jwsdp-1.3\jwstutorial13\doc\JavaWSTutorial.pdf**
6	Copiar Aplicación en Tomcat Server	Descompactar **D:\Application_Toolkit\version producción\ddmmyyyy.zip** (por ejemplo, **30082004.zip**) En **C:\jwsdp-1.3\webapps** Nota: Se generará el directorio **C:\jwsdp-1.3\webapps\produccion**
7	Instalar MySQL Clients and Servers	Descompactar **D:\Application_Toolkit\mysql\mysql-4.0.17-win.zip** En **C:\temp** Ejecutar **C:\temp\setup.exe** (3min) Borrar **C:\temp*.*** Ejecutar **C:\mysql\bin\mysqld-nt.exe** Ejecutar **C:\mysql\bin\winmysqladmin.exe** Escribir Username: **admin** Password : **admin.**
8	Copiar la base de datos en mysql	Copiar **C:\jwsdp-1.3\webapps\produccion\tesis\produccion.sql** En **C:\mysql\bin**
9	Instalar la base de datos en mysql	Ir a ventana Command Posicionarse en **C:\mysql\bin** Ejecutar **C:\mysql\bin>mysql --user=root < produccion.sql**

10	Agregar classes de jdbc de mysql en Tomcat Server y en el Java 2 SDK	Descompactar **D:\Application_Toolkit\mysql\mysql-connector-java-3.0.10-stable.zip** En **C:\temp** Copiar **C:\temp\com y C:\temp\org (carpeta y contenido)** En **C:\jwsdp-1.3\common\classes** Y también en **C:\j2sdk1.4.2\lib** Borrar **c:\temp**
11	Instalar MySQL ODBC (2 min)	Ejecutar **D:\Application_Toolkit\mysql\MyODBC-3.51.06.exe**
12	Iniciar Tomcat Server	
13	Registrar aplicación en Tomcat Server	Abrir Internet Explorer o Netscape Ir a http://localhost:8080/manager/install?path=/produccion&war=file:c:/jwsdp-1.3/webapps/produccion Después, ir a http://localhost:8080/manager/list Nota: Deberá aparecer la aplicación registrada en el Tomcat Server de la siguiente manera: **/produccion:running:0:c:\jwsdp-1.3\webapps\produccion**
14	Colocar al compilador de Java en PATH	Ir a Inicio/Mi PC/ Propiedades En Ventana “Propiedades del Sistema” Ir a “Opciones Avanzadas”/”Variables de Entorno” **PATH=..;c:\j2sdk1.4.2\bin**
15	Colocar las bibliotecas (libraries) de java en el CLASSPATH	Ir a Inicio/Mi PC/ Propiedades En Ventana “Propiedades del Sistema” Ir a “Opciones Avanzadas”/”Variables de Entorno” **CLASSPATH=C:\J2SDK1.4.2;C:\J2SDK1.4.2\JRE\LIB;.\;** **C:\JWSDP-1.3\COMMON\LIB\servlet-api.jar;** **C:\J2SDK1.4.2\lib;C:\jwsdp-1.3\saaj\lib\saaj-api.jar;**
16	Comprobar que las bibliotecas están bien instaladas	Ir a ventana Command Ir a **c:\jwsdp-1.3\webapps\produc~1\web-inf\classes** Compilar clases **c:\jwsdp-1.3\webapps\produc~1\web-inf\classes\>javac ServicioHTTP.java** No deberán generarse errores
17	Instalar MSSOAP Toolkit	Ejecutar **C:\jwsdp-1.3\webapps\produccion\cliente_access\SOAPSDK.EXE**
18	Poner un acceso directo del cliente en el escritorio	Poner un acceso directo de **E:\jwsdp-1.3\webapps\produccion\cliente_access\WSCLIENT.MDB**

<table>
<tr>
<td>19</td>
<td>Habilitar el puerto 80 para todas las aplicaciones del Tomcat Web Server.</td>
<td>a) Abrir el navegador en http://localhost:8080/admin/index.jsp. El usuario/password es admin/admin
b) En la barra de herramientas desplegar la rama Service (Java Web Services Developer Pack)
 

c) Crear un nuevo conector (Connector)
 

d) Aceptar los parámetros que se sugieren, y en el campo PortNumber, escribir 80
e) Guardar los cambios y verificar que ahora se tienen dos conectores, y que el puerto 80 es del tipo HTTP
 
</td>
</tr>
<tr>
<td>20</td>
<td>Reiniciar la PC</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Iniciar Tomcat Server y Abrir el Acceso directo a WSCLient.MDB</td>
<td>Iniciar la demostración de la aplicación</td>
</tr>
</table>

DesInstalación de Aplicación "Base de datos Mysql con Interfase Java Web Services. Caso de estudio: Recepción de reportes de ventas y órdenes de reparación de la industria automotriz"

1	Remover aplicación (No hacerlo, solo en casos extremos)	Abrir Internet Explorer o Netscape Ir a **`http://localhost:8080/manager/remove?path=/produccion`** Nota: El navegador mostrará el siguiente mensaje: **`OK – Undeployed application at context path /produccion`**
2	Remover la base de datos	Ir a ventana command Posicionarse en **`c:\mysql\bin`** Ejecutar **`C:\mysql\bin>mysql`** **`Mysql>drop database producción`** **`Mysql>exit`**

Anexo7. Ejemplos Estructurales de Mensajes XML SOAP

Índice de Documentos

XML SOAP Requests (Solicitudes de Información al Servicio Web)

AltaAsesor.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaAsesor>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfcAsesor>MAHE150980</rfcAsesor>
        <nombre>Erika Marchan Hernández</nombre>
      </AltaAsesor>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AltaInventario.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaInventario>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
      - <inventario>
          <serieVehiculo>0123456789ABCDEFG</serieVehiculo>
          <serieVehiculo>3KS1N423BSL895001</serieVehiculo>
          <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
        </inventario>
      </AltaInventario>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AltaCliente.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaCliente>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
        <estado>DF</estado>
        <clave>sr</clave>
        <nombre>Samuel</nombre>
        <paterno>Camacho</paterno>
        <materno>Garcia</materno>
        <razonsocial>Samuel Camacho García</razonsocial>
        <calle>Insurgentes Sur</calle>
        <numero>1325</numero>
        <colonia>San Angel</colonia>
        <municipio>Magdalena Contreras</municipio>
        <cp>01010</cp>
      - <telefonos>
        - <telefono>
            <tipodetelefono>compradorcasa</tipodetelefono>
            <lada>0155</lada>
            <numero>55129213</numero>
          </telefono>
        - <telefono>
            <tipodetelefono>compradorcel</tipodetelefono>
            <lada>04455</lada>
            <numero>52202612</numero>
          </telefono>
        - <telefono>
            <tipodetelefono>compradoroficina</tipodetelefono>
            <lada>0155</lada>
            <numero>55133300</numero>
          </telefono>
        </telefonos>
      </AltaCliente>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AltaOrden.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaOrden>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <ordenid>153228</ordenid>
        <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
        <rfcAsesor>MAHE150980</rfcAsesor>
        <tipodeoperacion>mant</tipodeoperacion>
        <tipodeorden>garantia</tipodeorden>
        <tipodeservicio>express</tipodeservicio>
        <apertura>2004-07-20T12:48:37</apertura>
        <cierre />
        <promesa>2004-07-21T12:48:37</promesa>
        <aseguradora>Financial Credit Insurance</aseguradora>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
        <placas>ASL042</placas>
        <km>1500</km>
      </AltaOrden>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AltaUnidad.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaUnidad>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
      </AltaUnidad>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

AltaVendedor.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <AltaVendedor>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfcVendedor>CAAA120859</rfcVendedor>
        <nombre>Alfredo Cadena Armenta</nombre>
      </AltaVendedor>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

BajaAsesor.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <BajaAsesor>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfcAsesor>MAHE150980</rfcAsesor>
      </BajaAsesor>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

BajaCliente.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <BajaCliente>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
      </BajaCliente>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

BajaUnidad.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <BajaUnidad>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
      </BajaUnidad>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

BajaVendedor.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <BajaVendedor>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <rfcVendedor>CAAA120859</rfcVendedor>
      </BajaVendedor>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

CancelaOrden.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <CancelaOrden>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <ordenid>153228</ordenid>
      </CancelaOrden>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

CancelaVenta.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <CancelaVenta>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
      </CancelaVenta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

CierraOrden.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <CierraOrden>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <ordenid>153228</ordenid>
        <cierre>2004-07-21T00:00:00</cierre>
      </CierraOrden>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ImportesOrden.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ImportesOrden>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <ordenid>153228</ordenid>
        <ing_obratall>5000</ing_obratall>
        <uti_obratall>3200</uti_obratall>
        <ing_refatall>800</ing_refatall>
        <uti_refatall>600</uti_refatall>
        <ing_obrahyp>0</ing_obrahyp>
        <uti_obrahyp>0</uti_obrahyp>
        <ing_refahyp>0</ing_refahyp>
        <uti_refahyp>0</uti_refahyp>
      </ImportesOrden>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ListaAsesores.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ListaAsesores>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
      </ListaAsesores>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ListaClientes.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ListaClientes>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
      </ListaClientes>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ListaVendedores.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ListaVendedores>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
      </ListaVendedores>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ReportaVenta.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ReportaVenta>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
        <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
        <tipoventa>vmenudeo</tipoventa>
        <tipopago>financiamiento</tipopago>
        <rfcVendedor>CAAA120859</rfcVendedor>
        <km>200</km>
        <fechaEntrega>2004-07-20T12:45:24</fechaEntrega>
        <term>12</term>
        <tasa>3.5</tasa>
        <pagoMensual>3500</pagoMensual>
        <serieCambio>1KP1M423AML895941</serieCambio>
        <kmCambio>120000</kmCambio>
      </ReportaVenta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Transferencia.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <Transferencia>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
        <serieVehiculo>3LS1M423ASL895941</serieVehiculo>
        <agenciaiddestino>M1320</agenciaiddestino>
      </Transferencia>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

XML SOAP Responses (Respuestas a las solicitudes de Información al Servicio Web)

ListaClientesRespuesta.xml

```
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ListaClientesRespuesta xmlns:m="http://www.unam.mx/ListaClientesRespuesta">
      - <cliente>
          <rfc_o_curp_cliente>AMNT921205</rfc_o_curp_cliente>
          <estado>NL</estado>
          <clave>negocios</clave>
          <nombre>america enterprises</nombre>
          <paterno>america enterprises</paterno>
          <materno>america enterprises</materno>
          <razonsocial>america enterprises</razonsocial>
          <calle>12</calle>
          <numero>12</numero>
          <colonia>aragon</colonia>
          <municipio>gustavo a madero</municipio>
          <cp>55221</cp>
        - <telefonos>
          - <telefono>
              <tipodetelefono>compradoroficina</tipodetelefono>
              <lada>0181</lada>
              <numero>83124414</numero>
            </telefono>
          + <telefono>
          </telefonos>
        </cliente>
      - <cliente>
          <rfc_o_curp_cliente>CAGS120271</rfc_o_curp_cliente>
          <estado>DF</estado>
          <clave>sr</clave>
          <nombre>Samuel</nombre>
          <paterno>Camacho</paterno>
          <materno>Garcia</materno>
          <razonsocial>Samuel Camacho Garcia</razonsocial>
          <calle>Insurgentes Sur</calle>
          <numero>1325</numero>
          <colonia>San Angel</colonia>
          <municipio>Magdalena Contreras</municipio>
          <cp>01010</cp>
        - <telefonos>
          + <telefono>
          + <telefono>
          - <telefono>
              <tipodetelefono>compradoroficina</tipodetelefono>
              <lada>0155</lada>
              <numero>55133300</numero>
            </telefono>
          </telefonos>
        </cliente>
      </m:ListaClientesRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ListaVendedoresRespuesta.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <ListaVendedores>
        <agenciaid>M1188</agenciaid>
        <password>moreno</password>
      </ListaVendedores>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ListaAsesoresRespuesta.xml

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ListaAsesoresRespuesta
        xmlns:m="http://www.unam.mx/ListaAsesoresRespuesta">
      - <asesor>
          <rfcAsesor>MAHE150980</rfcAsesor>
          <nombre>Erika Marchan Hernandez</nombre>
        </asesor>
      - <asesor>
          <rfcAsesor>RAIR760930</rfcAsesor>
          <nombre>Raul Ramirez Izquierdo</nombre>
        </asesor>
      </m:ListaAsesoresRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

ReportaVentaRespuesta.xml

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <m:ReportaVentaRespuesta
        xmlns:m="http://www.unam.mx/ReportaVentaRespuesta">
        <Respuesta>9999-ReportaVenta es Exitoso</Respuesta>
      </m:ReportaVentaRespuesta>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

SOAPFault5140.xml

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAP-ENV:Fault>
        <faultcode>SOAP-ENV:Server</faultcode>
        <faultactor>http://www.unam.mx/ReportaVenta</faultactor>
        <faultstring>5140 - El vehiculo 3LS1M423ASL895941 no ha sido
          dado de alta en el inventario. Utilice el metodo AltaUnidad o
          AltaInventario</faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

SOAPFault5170.xml

```
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAP-ENV:Fault>
        <faultcode>SOAP-ENV:Server</faultcode>
        <faultactor>http://www.unam.mx/ReportaVenta</faultactor>
        <faultstring>5170 - La venta del vehiculo 3LS1M423ASL895941
          ha sido previamente reportada por M1188. Contacte con
          esta agencia para mas informacion</faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Anexo 8. Web Services Description Language para la Interfaz

Índice de Documentos

```
<?xml version="1.0" ?>
- <definitions name="ordenes" targetNamespace="http://tempuri.org/wsdl/"
    xmlns:wsdlns="http://tempuri.org/wsdl/"
    xmlns:typens="http://tempuri.org/type"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <schema targetNamespace="http://tempuri.org/type"
        xmlns="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
        ENC="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified">
      - <xsd:complexType name="telefonos">
        - <xsd:all>
          - <xsd:element name="telefono" form="unqualified" minOccurs="1"
              maxOccurs="unbounded">
            - <xsd:complexType>
              - <xsd:sequence>
                  <xsd:element name="tipodetelefono" type="xsd:string"
                    form="unqualified" />
                  <xsd:element name="lada" type="xsd:unsignedLong"
                    form="unqualified" />
                  <xsd:element name="numero" type="xsd:unsignedLong"
                    form="unqualified" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:all>
        </xsd:complexType>
      - <xsd:complexType name="ListaAsesoresResultado">
        - <xsd:all>
          - <xsd:element name="asesor" form="unqualified" minOccurs="1"
              maxOccurs="unbounded">
            - <xsd:complexType>
              - <xsd:sequence>
                  <xsd:element name="rfcAsesor" type="xsd:string"
                    form="unqualified" />
                  <xsd:element name="nombre" type="xsd:string"
                    form="unqualified" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:all>
        </xsd:complexType>
      - <xsd:complexType name="ListaClientesResultado">
        - <xsd:all>
          - <xsd:element name="cliente" form="unqualified" minOccurs="1"
              maxOccurs="unbounded">
            - <xsd:complexType>
              - <xsd:sequence>
                  <xsd:element name="agenciaid" type="tns:styClave"
```

```
form="unqualified" />
  <xsd:element name="password" type="tns:styClave"
    form="unqualified" />
  <xsd:element name="rfc_o_curp_cliente"
    type="xsd:string" form="unqualified" />
  <xsd:element name="estado" type="xsd:string"
    form="unqualified" />
  <xsd:element name="clave" type="xsd:string"
    form="unqualified" />
  <xsd:element name="nombre" type="xsd:string"
    form="unqualified" />
  <xsd:element name="paterno" type="xsd:string"
    form="unqualified" />
  <xsd:element name="materno" type="xsd:string"
    form="unqualified" />
  <xsd:element name="razonsocial" type="xsd:string"
    form="unqualified" />
  <xsd:element name="calle" type="xsd:string"
    form="unqualified" />
  <xsd:element name="numero" type="xsd:string"
    form="unqualified" />
  <xsd:element name="colonia" type="xsd:string"
    form="unqualified" />
  <xsd:element name="municipio" type="xsd:string"
    form="unqualified" />
  <xsd:element name="cp" type="xsd:string"
    form="unqualified" />
- <xsd:element name="telefonos" form="unqualified">
  - <xsd:complexType>
    - <xsd:all>
      - <xsd:element name="telefono"
          form="unqualified" minOccurs="1"
          maxOccurs="unbounded">
        - <xsd:complexType>
          - <xsd:sequence>
              <xsd:element
                name="tipodetelefono"
                type="xsd:string"
                form="unqualified" />
              <xsd:element name="lada"
                type="xsd:unsignedLong"
                form="unqualified" />
              <xsd:element name="numero"
                type="xsd:unsignedLong"
                form="unqualified" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

```
            </xsd:element>
          </xsd:all>
        </xsd:complexType>
      </schema>
    </types>
- <message name="ordenesAltaAsesor">
      <part name="agenciaid" type="xsd:string" />
      <part name="password" type="xsd:string" />
      <part name="rfcAsesor" type="xsd:string" />
      <part name="nombre" type="xsd:string" />
    </message>
- <message name="ordenesAltaAsesorRespuesta">
      <part name="AltaAsesorRespuesta" type="xsd:string" />
    </message>
- <message name="ordenesBajaAsesor">
      <part name="agenciaid" type="xsd:string" />
      <part name="password" type="xsd:string" />
      <part name="rfcAsesor" type="xsd:string" />
    </message>
- <message name="ordenesBajaAsesorRespuesta">
      <part name="BajaAsesorRespuesta" type="xsd:string" />
    </message>
- <message name="ordenesListaAsesores">
      <part name="agenciaid" type="xsd:string" />
      <part name="password" type="xsd:string" />
    </message>
- <message name="ordenesListaAsesoresResultado">
      <part name="ListaAsesoresResultado"
        type="typens:ListaAsesoresResultado" />
    </message>
- <message name="ordenesAltaOrden">
      <part name="agenciaid" type="xsd:string" />
      <part name="password" type="xsd:string" />
      <part name="ordenid" type="xsd:string" />
      <part name="rfc_o_curp_cliente" type="xsd:string" />
      <part name="rfcAsesor" type="xsd:string" />
      <part name="tipodeoperacion" type="xsd:string" />
      <part name="tipodeorden" type="xsd:string" />
      <part name="tipodeservicio" type="xsd:string" />
      <part name="apertura" type="xsd:string" />
      <part name="cierre" type="xsd:string" />
      <part name="promesa" type="xsd:string" />
      <part name="aseguradora" type="xsd:string" />
      <part name="serieVehiculo" type="xsd:string" />
      <part name="placas" type="xsd:string" />
      <part name="km" type="xsd:string" />
    </message>
- <message name="ordenesAltaOrdenRespuesta">
      <part name="AltaOrdenRespuesta" type="xsd:string" />
    </message>
- <message name="ordenesImportesOrden">
      <part name="agenciaid" type="xsd:string" />
```

```
    <part name="password" type="xsd:string" />
    <part name="ordenid" type="xsd:string" />
    <part name="ing_obratall" type="xsd:string" />
    <part name="uti_obratall" type="xsd:string" />
    <part name="ing_refatall" type="xsd:string" />
    <part name="uti_refatall" type="xsd:string" />
    <part name="ing_obrahyp" type="xsd:string" />
    <part name="uti_obrahyp" type="xsd:string" />
    <part name="ing_refahyp" type="xsd:string" />
    <part name="uti_refahyp" type="xsd:string" />
  </message>
- <message name="ordenesImportesOrdenRespuesta">
    <part name="ImportesOrdenRespuesta" type="xsd:string" />
  </message>
- <message name="ordenesCierraOrden">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="ordenid" type="xsd:string" />
    <part name="cierre" type="xsd:string" />
  </message>
- <message name="ordenesCierraOrdenRespuesta">
    <part name="CierraOrdenRespuesta" type="xsd:string" />
  </message>
- <message name="ordenesCancelaOrden">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="ordenid" type="xsd:string" />
  </message>
- <message name="ordenesCancelaOrdenRespuesta">
    <part name="CancelaOrdenRespuesta" type="xsd:string" />
  </message>
- <message name="ordenesAltaCliente">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfc_o_curp_cliente" type="xsd:string" />
    <part name="estado" type="xsd:string" />
    <part name="clave" type="xsd:string" />
    <part name="nombre" type="xsd:string" />
    <part name="paterno" type="xsd:string" />
    <part name="materno" type="xsd:string" />
    <part name="razonsocial" type="xsd:string" />
    <part name="calle" type="xsd:string" />
    <part name="numero" type="xsd:string" />
    <part name="colonia" type="xsd:string" />
    <part name="municipio" type="xsd:string" />
    <part name="cp" type="xsd:string" />
    <part name="telefonos" type="typens:telefonos" />
  </message>
- <message name="ordenesAltaClienteRespuesta">
    <part name="AltaClienteRespuesta" type="xsd:string" />
  </message>
- <message name="ordenesBajaCliente">
```

```
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfc_o_curp_cliente" type="xsd:string" />
  </message>
- <message name="ordenesBajaClienteRespuesta">
    <part name="BajaClienteRespuesta" type="xsd:string" />
  </message>
- <message name="ordenesListaClientes">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
  </message>
- <message name="ordenesListaClientesResultado">
    <part name="ListaClientesResultado"
      type="typens:ListaClientesResultado" />
  </message>
- <portType name="ordenesSoapPort">
  - <operation name="AltaAsesor" parameterOrder="agenciaid password orden
      rfcAsesor nombre">
      <input message="wsdlns:ordenesAltaAsesor" />
      <output message="wsdlns:ordenesAltaAsesorRespuesta" />
    </operation>
  - <operation name="BajaAsesor" parameterOrder="agenciaid password
      rfcAsesor">
      <input message="wsdlns:ordenesBajaAsesor" />
      <output message="wsdlns:ordenesBajaAsesorRespuesta" />
    </operation>
  - <operation name="ListaAsesores" parameterOrder="agenciaid password">
      <input message="wsdlns:ordenesListaAsesores" />
      <output message="wsdlns:ordenesListaAsesoresRespuesta" />
    </operation>
  - <operation name="AltaOrden" parameterOrder="agenciaid password ordenid
      rfc_o_curp_cliente rfcAsesor tipodeoperacion tipodeorden tipodeservicio
      apertura cierre promesa aseguradora serieVehiculo placas km">
      <input message="wsdlns:ordenesAltaOrden" />
      <output message="wsdlns:ordenesAltaOrdenRespuesta" />
    </operation>
  - <operation name="ImportesOrden" parameterOrder="agenciaid password
      ordenid ing_obratall uti_obratall ing_refatall uti_refatall ing_obrahyp
      uti_obrahyp ing_refahyp uti_refahyp">
      <input message="wsdlns:ordenesImportesOrden" />
      <output message="wsdlns:ordenesImportesOrdenRespuesta" />
    </operation>
  - <operation name="CierraOrden" parameterOrder="agenciaid password ordenid
      cierre">
      <input message="wsdlns:ordenesCierraOrden" />
      <output message="wsdlns:ordenesCierraOrdenRespuesta" />
    </operation>
  - <operation name="CancelaOrden" parameterOrder="agenciaid password
      ordenid">
      <input message="wsdlns:ordenesCancelaOrden" />
      <output message="wsdlns:ordenesCancelaOrdenRespuesta" />
    </operation>
  - <operation name="AltaCliente" parameterOrder="agenciaid password
```

```
rfc_o_curp_cliente estado clave nombre paterno materno razonsocial calle
    numero colonia municipio cp telefonos">
    <input message="wsdlns:ordenesAltaCliente" />
    <output message="wsdlns:ordenesAltaClienteRespuesta" />
  </operation>
- <operation name="BajaCliente" parameterOrder="agenciaid password
    rfc_o_curp_cliente">
    <input message="wsdlns:ordenesBajaCliente" />
    <output message="wsdlns:ordenesBajaClienteRespuesta" />
  </operation>
- <operation name="ListaClientes" parameterOrder="agenciaid password">
    <input message="wsdlns:ordenesListaClientes" />
    <output message="wsdlns:ordenesListaClientesResultado" />
  </operation>
</portType>
- <binding name="ordenesSoapBinding" type="wsdlns:ordenesSoapPort">
  <stk:binding preferredEncoding="UTF-8" />
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="AltaAsesor">
    <soap:operation soapAction="http://www.unam.mx/AltaAsesor" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="BajaAsesor">
    <soap:operation soapAction="http://www.unam.mx/BajaAsesor" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="ListaAsesores">
    <soap:operation soapAction="http://www.unam.mx/ListaAsesores" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="AltaOrden">
```

```
    <soap:operation soapAction="http://www.unam.mx/AltaOrden" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="ImportesOrden">
    <soap:operation soapAction="http://www.unam.mx/ImportesOrden" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="CierraOrden">
    <soap:operation soapAction="http://www.unam.mx/CierraOrden" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="CancelaOrden">
    <soap:operation soapAction="http://www.unam.mx/CancelaOrden" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="AltaCliente">
    <soap:operation soapAction="http://www.unam.mx/AltaCliente" />
  - <input>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
      <soap:body use="encoded" namespace=""
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
```

```
    </operation>
  - <operation name="BajaCliente">
      <soap:operation soapAction="http://www.unam.mx/BajaCliente" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="ListaClientes">
      <soap:operation soapAction="http://www.unam.mx/ListaClientes" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
- <service name="ordenes">
  - <port name="ordenesSoapPort" binding="wsdlns:ordenesSoapBinding">
      <soap:address
        location="http://localhost:8080/produccion/servlet/ServicioHTTP" />
    </port>
  </service>
</definitions>
```

```
<?xml version="1.0" ?>
- <definitions name="ventas" targetNamespace="http://tempuri.org/wsdl/"
    xmlns:wsdlns="http://tempuri.org/wsdl/"
    xmlns:typens="http://tempuri.org/type"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <types>
    - <schema targetNamespace="http://tempuri.org/type"
        xmlns="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
        ENC="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        elementFormDefault="qualified">
      - <xsd:complexType name="inventario">
        - <xsd:all>
            <xsd:element name="serieVehiculo" type="tns:stySerieVehiculo"
              form="unqualified" minOccurs="1" maxOccurs="unbounded" />
          </xsd:all>
        </xsd:complexType>
      - <xsd:complexType name="telefonos">
        - <xsd:all>
          - <xsd:element name="telefono" form="unqualified" minOccurs="1"
              maxOccurs="unbounded">
            - <xsd:complexType>
              - <xsd:sequence>
                  <xsd:element name="tipodetelefono" type="xsd:string"
                    form="unqualified" />
                  <xsd:element name="lada" type="xsd:unsignedLong"
                    form="unqualified" />
                  <xsd:element name="numero" type="xsd:unsignedLong"
                    form="unqualified" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:all>
        </xsd:complexType>
      - <xsd:complexType name="ListaVendedoresResultado">
        - <xsd:all>
          - <xsd:element name="vendedor" form="unqualified" minOccurs="1"
              maxOccurs="unbounded">
            - <xsd:complexType>
              - <xsd:sequence>
                  <xsd:element name="rfcVendedor" type="xsd:string"
                    form="unqualified" />
                  <xsd:element name="nombre" type="xsd:string"
                    form="unqualified" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:all>
        </xsd:complexType>
      - <xsd:complexType name="ListaClientesResultado">
```

```
- <xsd:all>
  - <xsd:element name="cliente" form="unqualified" minOccurs="1"
      maxOccurs="unbounded">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="agenciaid" type="tns:styClave"
            form="unqualified" />
          <xsd:element name="password" type="tns:styClave"
            form="unqualified" />
          <xsd:element name="rfc_o_curp_cliente"
            type="xsd:string" form="unqualified" />
          <xsd:element name="estado" type="xsd:string"
            form="unqualified" />
          <xsd:element name="clave" type="xsd:string"
            form="unqualified" />
          <xsd:element name="nombre" type="xsd:string"
            form="unqualified" />
          <xsd:element name="paterno" type="xsd:string"
            form="unqualified" />
          <xsd:element name="materno" type="xsd:string"
            form="unqualified" />
          <xsd:element name="razonsocial" type="xsd:string"
            form="unqualified" />
          <xsd:element name="calle" type="xsd:string"
            form="unqualified" />
          <xsd:element name="numero" type="xsd:string"
            form="unqualified" />
          <xsd:element name="colonia" type="xsd:string"
            form="unqualified" />
          <xsd:element name="municipio" type="xsd:string"
            form="unqualified" />
          <xsd:element name="cp" type="xsd:string"
            form="unqualified" />
        - <xsd:element name="telefonos" form="unqualified">
          - <xsd:complexType>
            - <xsd:all>
              - <xsd:element name="telefono"
                  form="unqualified" minOccurs="1"
                  maxOccurs="unbounded">
                - <xsd:complexType>
                  - <xsd:sequence>
                      <xsd:element
                        name="tipodetelefono"
                        type="xsd:string"
                        form="unqualified" />
                      <xsd:element name="lada"
                        type="xsd:unsignedLong"
                        form="unqualified" />
                      <xsd:element name="numero"
                        type="xsd:unsignedLong"
                        form="unqualified" />
                    </xsd:sequence>
                  </xsd:complexType>
                </xsd:element>
```

```
                    </xsd:all>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:all>
      </xsd:complexType>
    </schema>
  </types>
- <message name="ventasAltaUnidad">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="serieVehiculo" type="xsd:string" />
  </message>
- <message name="ventasAltaUnidadRespuesta">
    <part name="AltaUnidadRespuesta" type="xsd:string" />
  </message>
- <message name="ventasBajaUnidad">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="serieVehiculo" type="xsd:string" />
  </message>
- <message name="ventasBajaUnidadRespuesta">
    <part name="BajaUnidadRespuesta" type="xsd:string" />
  </message>
- <message name="ventasAltaInventario">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="inventario" type="typens:inventario" />
  </message>
- <message name="ventasAltaInventarioRespuesta">
    <part name="AltaInventarioRespuesta" type="xsd:string" />
  </message>
- <message name="ventasAltaVendedor">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfcVendedor" type="xsd:string" />
    <part name="nombre" type="xsd:string" />
  </message>
- <message name="ventasAltaVendedorRespuesta">
    <part name="AltaVendedorRespuesta" type="xsd:string" />
  </message>
- <message name="ventasBajaVendedor">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfcVendedor" type="xsd:string" />
  </message>
- <message name="ventasBajaVendedorRespuesta">
    <part name="BajaVendedorRespuesta" type="xsd:string" />
  </message>
- <message name="ventasListaVendedores">
```

```
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
  </message>
- <message name="ventasListaVendedoresResultado">
    <part name="ListaVendedoresResultado"
      type="typens:ListaVendedoresResultado" />
  </message>
- <message name="ventasAltaCliente">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfc_o_curp_cliente" type="xsd:string" />
    <part name="estado" type="xsd:string" />
    <part name="clave" type="xsd:string" />
    <part name="nombre" type="xsd:string" />
    <part name="paterno" type="xsd:string" />
    <part name="materno" type="xsd:string" />
    <part name="razonsocial" type="xsd:string" />
    <part name="calle" type="xsd:string" />
    <part name="numero" type="xsd:string" />
    <part name="colonia" type="xsd:string" />
    <part name="municipio" type="xsd:string" />
    <part name="cp" type="xsd:string" />
    <part name="telefonos" type="typens:telefonos" />
  </message>
- <message name="ventasAltaClienteRespuesta">
    <part name="AltaClienteRespuesta" type="xsd:string" />
  </message>
- <message name="ventasBajaCliente">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="rfc_o_curp_cliente" type="xsd:string" />
  </message>
- <message name="ventasBajaClienteRespuesta">
    <part name="BajaClienteRespuesta" type="xsd:string" />
  </message>
- <message name="ventasListaClientes">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
  </message>
- <message name="ventasListaClientesResultado">
    <part name="ListaClientesResultado"
      type="typens:ListaClientesResultado" />
  </message>
- <message name="ventasReportaVenta">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="serieVehiculo" type="xsd:string" />
    <part name="rfc_o_curp_cliente" type="xsd:string" />
    <part name="tipoventa" type="xsd:string" />
    <part name="tipopago" type="xsd:string" />
    <part name="rfcVendedor" type="xsd:string" />
    <part name="km" type="xsd:string" />
```

```
    <part name="fechaEntrega" type="xsd:string" />
    <part name="term" type="xsd:string" />
    <part name="tasa" type="xsd:string" />
    <part name="pagoMensual" type="xsd:string" />
    <part name="serieCambio" type="xsd:string" />
    <part name="kmCambio" type="xsd:string" />
  </message>
- <message name="ventasReportaVentaRespuesta">
    <part name="ReportaVentaRespuesta" type="xsd:string" />
  </message>
- <message name="ventasCancelaVenta">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="serieVehiculo" type="xsd:string" />
  </message>
- <message name="ventasCancelaVentaRespuesta">
    <part name="CancelaVentaRespuesta" type="xsd:string" />
  </message>
- <message name="ventasTransferencia">
    <part name="agenciaid" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="serieVehiculo" type="xsd:string" />
    <part name="agenciaiddestino" type="xsd:string" />
  </message>
- <message name="ventasTransferenciaRespuesta">
    <part name="TransferenciaRespuesta" type="xsd:string" />
  </message>
- <portType name="ventasSoapPort">
  - <operation name="AltaUnidad" parameterOrder="agenciaid password
      serieVehiculo">
      <input message="wsdlns:ventasAltaUnidad" />
      <output message="wsdlns:ventasAltaUnidadRespuesta" />
    </operation>
  - <operation name="BajaUnidad" parameterOrder="agenciaid password
      serieVehiculo">
      <input message="wsdlns:ventasBajaUnidad" />
      <output message="wsdlns:ventasBajaUnidadRespuesta" />
    </operation>
  - <operation name="AltaInventario" parameterOrder="agenciaid password
      inventario">
      <input message="wsdlns:ventasAltaInventario" />
      <output message="wsdlns:ventasAltaInventarioRespuesta" />
    </operation>
  - <operation name="AltaVendedor" parameterOrder="agenciaid password
      rfcVendedor nombre">
      <input message="wsdlns:ventasAltaVendedor" />
      <output message="wsdlns:ventasAltaVendedorRespuesta" />
    </operation>
  - <operation name="BajaVendedor" parameterOrder="agenciaid password
      rfcVendedor">
      <input message="wsdlns:ventasBajaVendedor" />
      <output message="wsdlns:ventasBajaVendedorRespuesta" />
```

```
    </operation>
  - <operation name="ListaVendedores" parameterOrder="agenciaid password">
      <input message="wsdlns:ventasListaVendedores" />
      <output message="wsdlns:ventasListaVendedoresResultado" />
    </operation>
  - <operation name="AltaCliente" parameterOrder="agenciaid password
      rfc_o_curp_cliente estado clave nombre paterno materno razonsocial
      calle numero colonia municipio cp telefonos">
      <input message="wsdlns:ventasAltaCliente" />
      <output message="wsdlns:ventasAltaClienteRespuesta" />
    </operation>
  - <operation name="BajaCliente" parameterOrder="agenciaid password
      rfc_o_curp_cliente">
      <input message="wsdlns:ventasBajaCliente" />
      <output message="wsdlns:ventasBajaClienteRespuesta" />
    </operation>
  - <operation name="ListaClientes" parameterOrder="agenciaid password">
      <input message="wsdlns:ventasListaClientes" />
      <output message="wsdlns:ventasListaClientesResultado" />
    </operation>
  - <operation name="ReportaVenta" parameterOrder="agenciaid password
      serieVehiculo rfc_o_curp_cliente tipoventa tipopago rfcVendedor km
      fechaEntrega term tasa pagoMensual serieCambio kmCambio">
      <input message="wsdlns:ventasReportaVenta" />
      <output message="wsdlns:ventasReportaVentaRespuesta" />
    </operation>
  - <operation name="CancelaVenta" parameterOrder="agenciaid password
      serieVehiculo">
      <input message="wsdlns:ventasCancelaVenta" />
      <output message="wsdlns:ventasCancelaVentaRespuesta" />
    </operation>
  - <operation name="Transferencia" parameterOrder="agenciaid password
      serieVehiculo agenciaiddestino">
      <input message="wsdlns:ventasTransferencia" />
      <output message="wsdlns:ventasTransferenciaRespuesta" />
    </operation>
  </portType>
- <binding name="ventasSoapBinding" type="wsdlns:ventasSoapPort">
    <stk:binding preferredEncoding="UTF-8" />
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
  - <operation name="AltaUnidad">
      <soap:operation soapAction="http://www.unam.mx/AltaUnidad" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="BajaUnidad">
```

```
      <soap:operation soapAction="http://www.unam.mx/BajaUnidad" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="AltaInventario">
      <soap:operation soapAction="http://www.unam.mx/AltaInventario" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="AltaVendedor">
      <soap:operation soapAction="http://www.unam.mx/AltaVendedor" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="BajaVendedor">
      <soap:operation soapAction="http://www.unam.mx/BajaVendedor" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="ListaVendedores">
      <soap:operation soapAction="http://www.unam.mx/ListaVendedores" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
```

```
    </operation>
  - <operation name="AltaCliente">
      <soap:operation soapAction="http://www.unam.mx/AltaCliente" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="BajaCliente">
      <soap:operation soapAction="http://www.unam.mx/BajaCliente" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="ListaClientes">
      <soap:operation soapAction="http://www.unam.mx/ListaClientes" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="ReportaVenta">
      <soap:operation soapAction="http://www.unam.mx/ReportaVenta" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="CancelaVenta">
      <soap:operation soapAction="http://www.unam.mx/CancelaVenta" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
```

```
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  - <operation name="Transferencia">
      <soap:operation soapAction="http://www.unam.mx/Transferencia" />
    - <input>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
    - <output>
        <soap:body use="encoded" namespace=""
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
- <service name="ventas">
  - <port name="ventasSoapPort" binding="wsdlns:ventasSoapBinding">
      <soap:address
        location="http://localhost:8080/produccion/servlet/ServicioHTTP" />
    </port>
  </service>
</definitions>
```

Anexo 9. XML Schemas para los XML SOAP Requests

Índice de Documentos

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaAsesor" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfcAsesor" type="tns:styRFC"
                  form="unqualified" />
                <xs:element name="nombre" type="tns:styNombre"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaCliente" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfc_o_curp_cliente" type="xs:string"
                  form="unqualified" />
                <xs:element name="estado" type="xs:string"
                  form="unqualified" />
                <xs:element name="clave" type="xs:string"
                  form="unqualified" />
                <xs:element name="nombre" type="xs:string"
                  form="unqualified" />
                <xs:element name="paterno" type="xs:string"
                  form="unqualified" />
                <xs:element name="materno" type="xs:string"
                  form="unqualified" />
                <xs:element name="razonsocial" type="xs:string"
                  form="unqualified" />
                <xs:element name="calle" type="xs:string"
                  form="unqualified" />
                <xs:element name="numero" type="xs:string"
                  form="unqualified" />
                <xs:element name="colonia" type="xs:string"
                  form="unqualified" />
                <xs:element name="municipio" type="xs:string"
                  form="unqualified" />
                <xs:element name="cp" type="xs:string"
                  form="unqualified" />
              - <xs:element name="telefonos" form="unqualified">
                - <xs:complexType>
                  - <xs:all>
                    - <xs:element name="telefono" form="unqualified"
                        minOccurs="1" maxOccurs="unbounded">
                      - <xs:complexType>
                        - <xs:sequence>
```

```
                        <xs:element name="tipodetelefono"
                          type="xs:string"
                          form="unqualified" />
                        <xs:element name="lada"
                          type="xs:unsignedLong"
                          form="unqualified" />
                        <xs:element name="numero"
                          type="xs:unsignedLong"
                          form="unqualified" />
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:all>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaInventario" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
              - <xs:element name="inventario" form="unqualified">
                - <xs:complexType>
                  - <xs:all>
                      <xs:element name="serieVehiculo"
                        type="tns:stySerieVehiculo"
                        form="unqualified" minOccurs="1"
                        maxOccurs="unbounded" />
                    </xs:all>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaOrden" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="ordenid" type="xs:string"
                  form="unqualified" />
                <xs:element name="rfc_o_curp_cliente" type="xs:string"
                  form="unqualified" />
                <xs:element name="rfcAsesor" type="tns:styRFC"
                  form="unqualified" />
                <xs:element name="tipodeoperacion" type="xs:string"
                  form="unqualified" />
                <xs:element name="tipodeorden" type="xs:string"
                  form="unqualified" />
                <xs:element name="tipodeservicio" type="xs:string"
                  form="unqualified" />
                <xs:element name="apertura" type="xs:dateTime"
                  form="unqualified" />
                <xs:element name="cierre" type="xs:string"
                  form="unqualified" />
                <xs:element name="promesa" type="xs:dateTime"
                  form="unqualified" />
                <xs:element name="aseguradora" type="xs:string"
                  form="unqualified" />
                <xs:element name="serieVehiculo" type="xs:string"
                  form="unqualified" />
                <xs:element name="placas" type="xs:string"
                  form="unqualified" />
                <xs:element name="km" type="xs:unsignedLong"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
```

```
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaUnidad" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="serieVehiculo"
                  type="tns:stySerieVehiculo" form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="AltaVendedor" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfcVendedor" type="tns:styRFC"
                  form="unqualified" />
                <xs:element name="nombre" type="tns:styNombre"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="BajaAsesor" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfcAsesor" type="tns:styRFC"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="BajaCliente" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfc_o_curp_cliente" type="xs:string"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="BajaUnidad" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="serieVehiculo"
                  type="tns:stySerieVehiculo" form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="BajaVendedor" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="rfcVendedor" type="tns:styRFC"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="CancelaOrden" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="ordenid" type="xs:string"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="CancelaVenta" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="serieVehiculo"
                  type="tns:stySerieVehiculo" form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="CierraOrden" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="ordenid" type="xs:string"
                  form="unqualified" />
                <xs:element name="cierre" type="xs:dateTime"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="ImportesOrden" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="ordenid" type="xs:string"
                  form="unqualified" />
                <xs:element name="ing_obratall" type="xs:float"
                  form="unqualified" />
                <xs:element name="uti_obratall" type="xs:float"
                  form="unqualified" />
                <xs:element name="ing_refatall" type="xs:float"
                  form="unqualified" />
                <xs:element name="uti_refatall" type="xs:float"
                  form="unqualified" />
                <xs:element name="ing_obrahyp" type="xs:float"
                  form="unqualified" />
                <xs:element name="uti_obrahyp" type="xs:float"
                  form="unqualified" />
                <xs:element name="ing_refahyp" type="xs:float"
                  form="unqualified" />
                <xs:element name="uti_refahyp" type="xs:float"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="ListaAsesores" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="ListaClientes" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="ListaVendedores" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="ReportaVenta" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="serieVehiculo"
                  type="tns:stySerieVehiculo" form="unqualified" />
                <xs:element name="rfc_o_curp_cliente" type="xs:string"
                  form="unqualified" />
                <xs:element name="tipoventa" type="xs:string"
                  form="unqualified" />
                <xs:element name="tipopago" type="xs:string"
                  form="unqualified" />
                <xs:element name="rfcVendedor" type="tns:styRFC"
                  form="unqualified" />
                <xs:element name="km" type="xs:unsignedLong"
                  form="unqualified" />
                <xs:element name="fechaEntrega" type="xs:dateTime"
                  form="unqualified" />
                <xs:element name="term" type="xs:unsignedLong"
                  form="unqualified" />
                <xs:element name="tasa" type="xs:float"
                  form="unqualified" />
                <xs:element name="pagoMensual" type="xs:float"
                  form="unqualified" />
                <xs:element name="serieCambio" type="xs:string"
                  form="unqualified" />
                <xs:element name="kmCambio" type="xs:unsignedLong"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
  - <xs:simpleType name="stySerieVehiculo">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-zA-Z]){17}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styClave">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([0-9]|[a-zA-Z]){5,}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styRFC">
    - <xs:restriction base="xs:string">
        <xs:pattern value="[a-zA-Z]{4}[0-9]{6}(.){0,10}" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:simpleType name="styNombre">
    - <xs:restriction base="xs:string">
        <xs:pattern value="([a-zA-Z]|(\s)){5,50}" />
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
    <xs:include schemaLocation="TiposSMM.xsd" />
  - <xs:element name="Envelope">
    - <xs:complexType>
      - <xs:sequence>
          <xs:element ref="tns:Body" minOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  - <xs:element name="Body">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element name="Transferencia" form="unqualified">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="agenciaid" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="password" type="tns:styClave"
                  form="unqualified" />
                <xs:element name="serieVehiculo"
                  type="tns:stySerieVehiculo" form="unqualified" />
                <xs:element name="agenciaiddestino" type="tns:styClave"
                  form="unqualified" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Anexo 10. Ejemplo de desarrollo en Arquitectura CORBA

1 & 2 Writing the Interface Definition
Before you start working with Java IDL, you need to install version 1.4 of J2SE. J2SE v.1.4 provides the Application Programming Interface (API) and Object Request Broker (ORB) needed to enable CORBA-based distributed object interaction, as well as the idlj compiler. The idlj compiler uses the IDL-to-Java language mapping to convert IDL interface definitions to corresponding Java interfaces, classes, and methods, which you can then use to implement your client and server code.

This section teaches you how to write a simple IDL interface definition and how to translate the IDL interface to Java. It also describes the purpose of each file generated by the `idlj` compiler.

These topics are included in this section:

Writing `Hello.idl`
To create the `Hello.idl` file,

1. Create a new directory, named `Hello`, for this application.
2. Start your favorite text editor and create a file named `Hello.idl` in this directory.
3. In your file, enter the code for the interface definition, **Hello.idl**:

```
4.  module HelloApp
5.  {
6.    interface Hello
7.    {
8.      string sayHello();
9.      oneway void shutdown();
10.   };
11. };
```

12. Save the file.

Understanding the IDL file

OMG IDL is the language used to describe the interfaces that client objects call and object implementations provide. An interface definition written in OMG IDL completely defines the interface and fully specifies each operation's parameters. An OMG IDL interface provides the information needed to develop clients that use the interface's operations.

Clients are written in languages for which mappings from OMG IDL concepts have been defined. The mapping of an OMG IDL concept to a client language construct will depend on the facilities available in the client language. OMG specifies a mapping from IDL to several different programming languages, including C, C++, Smalltalk, COBOL, Ada, Lisp, Python, and Java. When mapped, each statement in OMG IDL is translated to a corresponding statement in the programming language of choice.

For example, you could use the tool `idlj` to map an IDL interface to Java and implement the client class. When you mapped the same IDL to C++ and implemented the server in that language, the Java client (through the Java ORB) and C++ server (through the C++ ORB) interoperate as though they were written in the same language.
The IDL for "Hello World" is extremely simple; its single interface has but two operations. You need perform only three steps:

1. Declare the CORBA IDL module
2. Declare the interface
3. Declare the operations

Declaring the CORBA IDL Module

A CORBA module is a namespace that acts as a container for related interfaces and declarations. It corresponds closely to a Java package. Each module statement in an IDL file is mapped to a Java package statement.
The module statement looks like this:

```
module HelloApp
{
    // Subsequent lines of code here.
};
```

When you compile the IDL, the module statement will generate a package statement in the Java code.

Declaring the Interface

Like Java interfaces, CORBA interfaces declare the API contract an object has with other objects. Each interface statement in the IDL maps to a Java interface statement when mapped.
In your `Hello.idl` file, the interface statement looks like this:

```
module HelloApp
{
  interface Hello  // These lines
  {                // declare the
                   // interface
  };               // statement.
};
```

When you compile the IDL, this statement will generate an interface statement in the Java code.

Declaring the Operations

CORBA operations are the behavior that servers promise to perform on behalf of clients that invoke them. Each operation statement in the IDL generates a corresponding method statement in the generated Java interface.
In your `Hello.idl` file, the operation statement looks like this:

```
module HelloApp
{
  interface Hello
  {
    string sayHello();         // This line is an operation statement.
    oneway void shutdown();    // This line is another
  };
};
```

The interface definition for our little "Hello World" application is now complete.

Mapping `Hello.idl` to Java

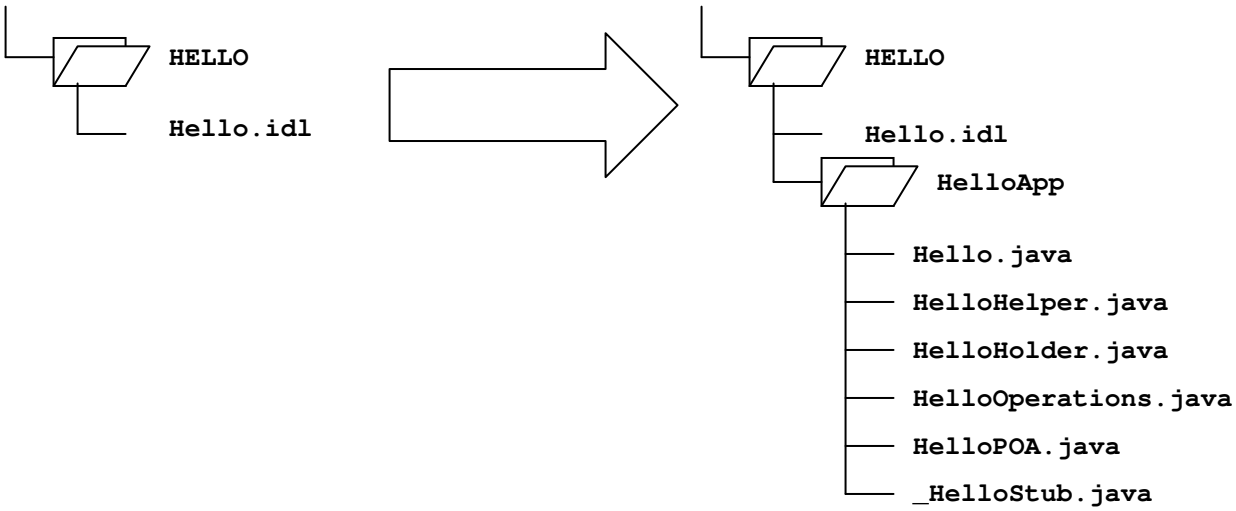
The tool `idlj` reads OMG IDL files and creates the required Java files. The `idlj` compiler defaults to generating only the client-side bindings. If you need both client-side bindings and server-side skeletons (as you do for our "Hello World" program), you must use the **`-fall`** option when running the `idlj` compiler. For more information on the IDL-to-Java compiler options, follow the link.
New in J2SE v.1.4: The default server-side mapping generated when either the `-fall` or `-fserver` arguments are used conform to Chapter 11, *Portable Object Adapter* (POA) of the CORBA 2.3.1 Specification (formal/99-10-07). For more information on the POA, link to Portable Object Adapter.
The advantages of using the Portable Object Adaptor (POA) are:

- Allow programmers to construct object implementations that are portable between different ORB products.
- Provide support for objects with persistent identities.

- Provide support for transparent activation of objects.
- Allow a single servant to support multiple object identities simultaneously.

1. Make sure that the `j2sdk/bin` directory (or the directory containing `idlj`, `java`, `javac`, and `orbd`) are in your path.
2. Go to a command line prompt.
3. Change to the directory containing your `Hello.idl` file.
4. Enter the compiler command:
5.
6.

```
idlj -fall Hello.idl
```



If you list the contents of the directory, you will see that a directory called `HelloApp` has been created and that it contains six files. Open `Hello.java` in your text editor. `Hello.java` is the *signature interface* and is used as the signature type in method declarations when interfaces of the specified type are used in other interfaces. It looks like this:

```
//Hello.java
package HelloApp;


/**
* HelloApp/Hello.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
* from Hello.idl
*/

public interface Hello extends HelloOperations, org.omg.CORBA.Object,
org.omg.CORBA.portable.IDLEntity
{
} // interface Hello
```

With an interface this simple, it is easy to see how the IDL statements map to the generated Java statements.

IDL Statement	Java Statement
`module HelloApp`	`Package HelloApp;`
`interface Hello`	`public interface Hello`

The single surprising item is the `extends` statement. All CORBA objects are derived from `org.omg.CORBA.Object` to ensure required CORBA functionality. The required code is generated by `idlj`; you do not need to do any mapping yourself.

In previous versions of the `idlj` compiler (known as `idltojava`), the operations defined on the IDL interface would exist in this file as well. Starting with J2SDK v1.3.0, in conformance with the CORBA 2.3.1 Specification (formal/99-10-07), the IDL-to-Java mapping puts all of the operations defined on the IDL interface in the *operations interface*, `HelloOperations.java`. The

operations interface is used in the server-side mapping and as a mechanism for providing optimized calls for co-located clients and servers. For `Hello.idl`, this file looks like this:

```
//HelloOperations.java
package HelloApp;


/**
* HelloApp/HelloOperations.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
* from Hello.idl
*/

public interface HelloOperations
{
  String sayHello ();
  void Shutdown ();
} // interface HelloOperations
```

Because there are only two operations defined in this interface, it is easy to see how the IDL statements map to the generated Java statements.

IDL Statement	Java Statement
`string sayHello();`	`String sayHello();`
`oneway void shutdown();`	`void Shutdown ();`

Understanding the `idlj` Compiler Output

The `idlj` compiler generates a number of files. The actual number of files generated depends on the options selected when the IDL file is compiled. The generated files provide standard functionality, so you can ignore them until it is time to deploy and run your program. Under J2SE v.1.4, the files generated by the `idlj` compiler for `Hello.idl`, with the **`-fall`** command line option, are:

- `HelloPOA.java`
  This abstract class is the stream-based server skeleton, providing basic CORBA functionality for the server. It extends `org.omg.PortableServer.Servant`, and implements the `InvokeHandler` interface and the `HelloOperations` interface. The server class, `HelloServant`, extends `HelloPOA`.
- `_HelloStub.java`
  This class is the client stub, providing CORBA functionality for the client. It extends `org.omg.CORBA.portable.ObjectImpl` and implements the `Hello.java` interface.
- `Hello.java`
  This interface contains the Java version of our IDL interface. The `Hello.java` interface extends `org.omg.CORBA.Object`, providing standard CORBA object functionality. It also extends the `HelloOperations` interface and `org.omg.CORBA.portable.IDLEntity`.
- `HelloHelper.java`
  This class provides auxiliary functionality, notably the `narrow()` method required to cast CORBA object references to their proper types. The Helper class is responsible for reading and writing the data type to CORBA streams, and inserting and extracting the data type from `Anys`. The Holder class delegates to the methods in the Helper class for reading and writing.
- `HelloHolder.java`
  This final class holds a public instance member of type `Hello`. Whenever the IDL type is an `out` or an `inout` parameter, the Holder class is used. It provides operations for `org.omg.CORBA.portable.OutputStream` and `org.omg.CORBA.portable.InputStream` arguments, which CORBA allows, but which do not map easily to Java's semantics. The Holder class delegates to the methods in the Helper class for reading and writing. It implements `org.omg.CORBA.portable.Streamable`.
- `HelloOperations.java`

This interface contains the methods `sayHello()` and `shutdown()`. The IDL-to-Java mapping puts all of the operations defined on the IDL interface into this file, which is shared by both the stubs and skeletons.

When you write the IDL interface, you do all the programming required to generate all these files for your distributed application. The next steps are to implement the client and server classes. In the steps that follow, you will create the `HelloClient.java` client class and the `HelloServer.java` server class.

Troubleshooting

- Error Message: "idlj" not found
  If you try to run `idlj` on the file `Hello.idl` and the system cannot find `idlj`, it is most likely not in your path. Make certain that the location of `idlj` (the J2SDK v.1.4 `bin` directory) is in your path, and try again.

3 Developing the Hello World Server

The example server consists of two classes, the servant and the server. The servant, `HelloImpl`, is the implementation of the `Hello` IDL interface; each `Hello` instance is implemented by a `HelloImpl` instance. The servant is a subclass of `HelloPOA`, which is generated by the `idlj` compiler from the example IDL.

The servant contains one method for each IDL operation, in this example, the `sayHello()` and `shutdown()` methods. Servant methods are just like ordinary Java methods; the extra code to deal with the ORB, with marshaling arguments and results, and so on, is provided by the skeleton.

The server class has the server's `main()` method, which:

- Creates and initializes an ORB instance
- Gets a reference to the root POA and activates the `POAManager`
- Creates a servant instance (the implementation of one CORBA `Hello` object) and tells the ORB about it
- Gets a CORBA object reference for a naming context in which to register the new CORBA object
- Gets the root naming context
- Registers the new object in the naming context under the name "Hello"
- Waits for invocations of the new object from the client

This lesson introduces the basics of writing a CORBA server. For an example of the "Hello World" program with a persistent object server, see *Example 2: Hello World with Persistent State*. For more discussion of CORBA servers, see *Developing Servers*.
The steps in this lesson cover:

1. Creating HelloServer.java
2. Understanding HelloServer.java
3. Compiling the Hello World Server

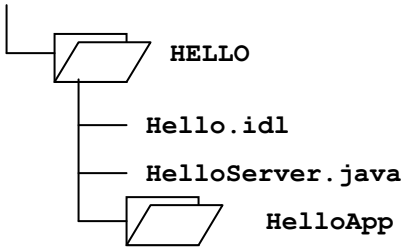
Creating HelloServer.java

To create `HelloServer.java`,
Start your text editor and create a file named `HelloServer.java` in your main project directory, `Hello`.
Enter the following code for `HelloServer.java` in the text file. The following section, Understanding HelloServer.java, explains each line of code in some detail.

```
1.    // HelloServer.java
2.    // Copyright and License
3.    import HelloApp.*;
4.    import org.omg.CosNaming.*;
5.    import org.omg.CosNaming.NamingContextPackage.*;
6.    import org.omg.CORBA.*;
7.    import org.omg.PortableServer.*;
8.    import org.omg.PortableServer.POA;
9.
10.   import java.util.Properties;
11.
12.   class HelloImpl extends HelloPOA {
13.     private ORB orb;
14.
15.     public void setORB(ORB orb_val) {
16.       orb = orb_val;
17.     }
```

```
18.
19.     // implement sayHello() method
20.     public String sayHello() {
21.       return "\nHello world !!\n";
22.     }
23.
24.     // implement shutdown() method
25.     public void shutdown() {
26.       orb.shutdown(false);
27.     }
28.   }
29.
30.
31.   public class HelloServer {
32.
33.     public static void main(String args[]) {
34.       try{
35.         // create and initialize the ORB
36.         ORB orb = ORB.init(args, null);
37.
38.         // get reference to rootpoa & activate the POAManager
39.         POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
40.         rootpoa.the_POAManager().activate();
41.
42.         // create servant and register it with the ORB
43.         HelloImpl helloImpl = new HelloImpl();
44.         helloImpl.setORB(orb);
45.
46.         // get object reference from the servant
47.         org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
48.         Hello href = HelloHelper.narrow(ref);
49.
50.         // get the root naming context
51.         org.omg.CORBA.Object objRef =
52.             orb.resolve_initial_references("NameService");
53.         // Use NamingContextExt which is part of the Interoperable
54.         // Naming Service (INS) specification.
55.         NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
56.
57.         // bind the Object Reference in Naming
58.         String name = "Hello";
59.         NameComponent path[] = ncRef.to_name( name );
60.         ncRef.rebind(path, href);
61.
62.         System.out.println("HelloServer ready and waiting ...");
63.
64.         // wait for invocations from clients
65.         orb.run();
66.       }
67.
68.         catch (Exception e) {
69.           System.err.println("ERROR: " + e);
70.           e.printStackTrace(System.out);
71.         }
72.
73.         System.out.println("HelloServer Exiting ...");
74.
75.     }
76.   }
```

Save and close HelloServer.java.



Understanding HelloServer.java

This section explains each line of `HelloServer.java`, describing what the code does, as well as why it is needed for this application.

Performing Basic Setup

The structure of a CORBA server program is the same as most Java applications: You import required library packages, declare the server class, define a `main()` method, and handle exceptions.

Importing Required Packages

First, we import the packages required for the server class:

```
// The package containing our stubs
import HelloApp.*;

// HelloServer will use the naming service
import org.omg.CosNaming.*;

// The package containing special exceptions thrown by the name service
import org.omg.CosNaming.NamingContextPackage.*;

// All CORBA applications need these classes
import org.omg.CORBA.*;

// Classes needed for the Portable Server Inheritance Model
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

// Properties to initiate the ORB
import java.util.Properties;
```

Defining the Servant Class

In this example, we are defining the class for the servant object within `HelloServer.java`, but outside the `HelloServer` class.

```
class HelloImpl extends HelloPOA
{
  // The sayHello() and shutdown() methods go here.
}
```

The servant is a subclass of `HelloPOA` so that it inherits the general CORBA functionality generated for it by the compiler.

First, we create a private variable, `orb` that is used in the `setORB(ORB)` method. The `setORB` method is a private method defined by the application developer so that they can set the ORB value with the servant. This ORB value is used to invoke `shutdown()` on that specific ORB in response to the `shutdown()` method invocation from the client.

```
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }
```

Next, we declare and implement the required `sayHello()` method:

```
  public String sayHello()
  {
    return "\nHello world!!\n";
  }
```

And last of all, we implement the `shutdown()` method in a similar way. The `shutdown()` method calls the `org.omg.CORBA.ORB.shutdown(boolean)` method for the ORB. The `shutdown(false)` operation indicate that the ORB should shut down immediately, without waiting for processing to complete.

```
  public void shutdown() {
    orb.shutdown(false);
  }
```

Declaring the Server Class

The next step is to declare the server class:

```
public class HelloServer
{
  // The main() method goes here.
}
```

Defining the main() Method

Every Java application needs a `main` method. It is declared within the scope of the `HelloServer` class:

```
  public static void main(String args[])
  {
    // The try-catch block goes here.
  }
```

Handling CORBA System Exceptions

Because all CORBA programs can throw CORBA system exceptions at runtime, all of the `main()` functionality is placed within a try-catch block. CORBA programs throw runtime exceptions whenever trouble occurs during any of the processes (marshaling, unmarshaling, upcall) involved in invocation. The exception handler simply prints the exception and its stack trace to standard output so you can see what kind of thing has gone wrong.

The try-catch block is set up inside `main()`, as shown:

```
    try{

      // The rest of the HelloServer code goes here.

    } catch(Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }
```

Creating and Initializing an ORB Object

A CORBA server needs a local ORB object, as does the CORBA client. Every server instantiates an ORB and registers its servant objects so that the ORB can find the server when it receives an invocation for it.

The ORB variable is declared and initialized inside the try-catch block.

```
      ORB orb = ORB.init(args, null);
```

The call to the ORB's `init()` method passes in the server's command line arguments, allowing you to set certain properties at runtime.

Get a Reference to the Root POA and Activate the `POAManager`

The ORB obtains the initial object references to services such as the Name Service using the method `resolve_initial_references`.

The reference to the root POA is retrieved and the `POAManager` is activated from within the try-catch block.

```
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
```

The `activate()` operation changes the state of the POA manager to active, causing associated POAs to start processing requests. The POA manager encapsulates the processing state of the POAs with which it is associated. Each `POA` object has an associated `POAManager` object. A POA manager may be associated with one or more POA objects.

Managing the Servant Object

A server is a process that instantiates one or more servant objects. The servant inherits from the interface generated by `idlj` and actually performs the work of the operations on that interface. Our `HelloServer` needs a `HelloImpl`.

Instantiating the Servant Object

We instantiate the servant object inside the try-catch block, just after activating the POA manager, as shown:

```
      HelloImpl helloImpl = new HelloImpl();
```

The section of code describing the servant class was explained previously.

In the next line of code, `setORB(orb)` is defined on the servant so that `ORB.shutdown()` can be called as part of the shutdown operation. This step is required because of the `shutdown()` method defined in `Hello.idl`.

```
helloImpl.setORB(orb);
```

There are other options for implementing the shutdown operation. In this example, the `shutdown()` method called on the `Object` takes care of shutting down an ORB. In another implementation, the shutdown method implementation could have simply set a flag, which the server could have checked and called `shutdown()`.

The next set of code is used to get the object reference associated with the servant. The `narrow()` method is required to cast CORBA object references to their proper types.

```
org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
Hello href = HelloHelper.narrow(ref);
```

Working with COS Naming

The `HelloServer` works with the Common Object Services (COS) Naming Service to make the servant object's operations available to clients. The server needs an object reference to the naming service so that it can publish the references to the objects implementing various interfaces. These object references are used by the clients for invoking methods. Another way a servant can make the objects available to clients for invocations is by stringifying the object references to a file.

The two options for Naming Services shipped with J2SE v.1.4 are:

- `orbd`, which includes both a Transient Naming Service and a Persistent Naming Service, in addition to a Server Manager.
- `tnameserv` - a Transient Naming Service.

This example uses `orbd`.

Obtaining the Initial Naming Context

In the try-catch block, below getting the object reference for the servant, we call `orb.resolve_initial_references()` to get an object reference to the name server:

```
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
```

The string "NameService" is defined for all CORBA ORBs. When you pass in that string, the ORB returns a naming context object that is an object reference for the name service. The string "NameService" indicates:

- The naming service will be persistent when using ORBD's naming service, as we do in this example.
- The naming service will be transient when using `tnameserv`.

The proprietary string "TNameService" indicates that the naming service will be transient when using ORBD's naming service.

Narrowing the Object Reference

As with all CORBA object references, `objRef` is a generic CORBA object. To use it as a `NamingContextExt object`, you must narrow it to its proper type. The call to `narrow()` is just below the previous statement:

```
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

Here you see the use of an `idlj`-generated helper class, similar in function to `HelloHelper`. The `ncRef` object is now an `org.omg.CosNaming.NamingContextExt` and you can use it to access the naming service and register the server, as shown in the next topic.

The `NamingContextExt object` is new to J2SE v.1.4, and is part of the Interoperable Naming Service specification.

Registering the Servant with the Name Server

Just below the call to `narrow()`, we create a new `NameComponent` array. Because the path to `Hello` has a single element, we create the single-element array that `NamingContext.resolve` requires for its work:

```
String name = "Hello";
NameComponent path[] = ncRef.to_name( name );
```

Finally, we pass `path` and the servant object to the naming service, binding the servant object to the "Hello" id:

```
ncRef.rebind(path, href);
```

Now, when the client calls `resolve("Hello")` on the initial naming context, the naming service returns an object reference to the `Hello` servant.

Waiting for Invocation

The previous sections describe the code that makes the server ready; the next section explains the code that enables it to simply wait around for a client to request its service. The following code, which is at the end of (but within) the try-catch block, shows how to accomplish this.

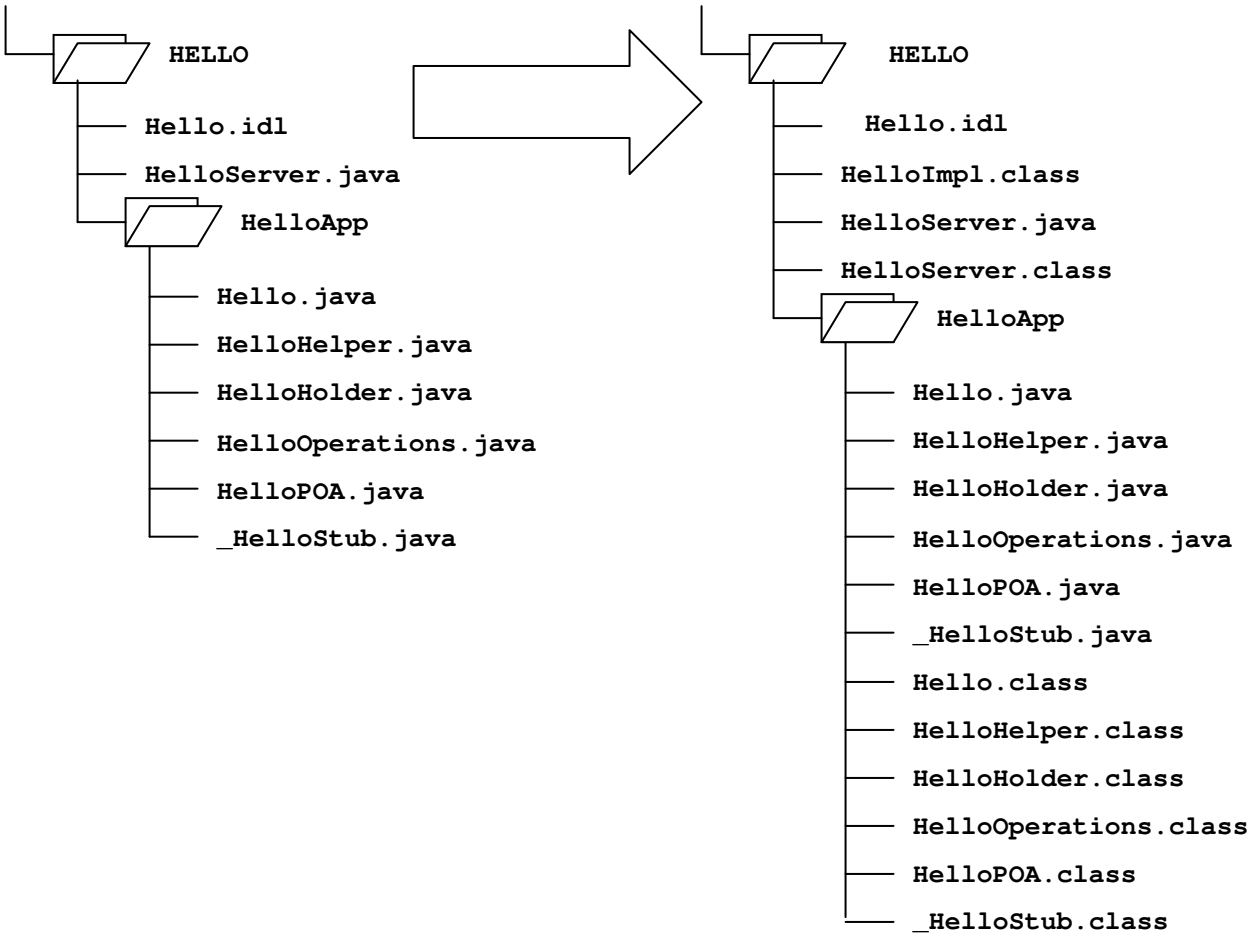
```
orb.run();
```

When called by the main thread, `ORB.run()` enables the ORB to perform work using the main thread, waiting until an invocation comes from the ORB. Because of its placement in `main()`, after an invocation completes and `sayHello()` returns, the server will wait again. This is the reason that the `HelloClient` explicitly shuts down the ORB after completing its task.

Compiling the Hello World Server

Now we will compile the `HelloServer.java` so that we can correct any errors before continuing with this tutorial.
Windows users note that you should substitute backslashes (\) for the slashes (/) in all paths in this document.
To compile `HelloServer.java`,

1. Change to the `Hello` directory.
2. Run the Java compiler on `HelloServer.java`:
3. `javac HelloServer.java HelloApp\*.java`
4. Correct any errors in your file and recompile if necessary.
5. The files `HelloServer.class` and `HelloImpl.class` are generated in the `Hello` directory.



Understanding The Server-Side Implementation Models

CORBA supports at least two different server-side mappings for implementing an IDL interface:

- The Inheritance Model
  Using the Inheritance Model, you implement the IDL interface using an implementation class that also extends the compiler-generated skeleton.
  Inheritance models include:
  - The OMG-standard, *POA*. Given an interface `My` defined in `My.idl`, the file `MyPOA.java` is generated by the `idlj` compiler. You must provide the implementation for `My` and it must inherit from `MyPOA`, a stream-based

skeleton that extends `org.omg.PortableServer.Servant`, which serves as the base class for all POA servant implementations.

New in J2SE v.1.4: The default server-side mapping generated when either the `-fall` or `-fserver` arguments are used conform to Chapter 11, *Portable Object Adapter* (POA) of the CORBA 2.3.1 Specification (formal/99-10-07). For more information on the POA, link to *Portable Object Adapter*.

The advantages of using the Portable Object Adaptor (POA) are:

- Allow programmers to construct object implementations that are portable between different ORB products.
- Provide support for objects with persistent identities.
- Provide support for transparent activation of objects.
- Allow a single servant to support multiple object identities simultaneously.

- *ImplBase*. Given an interface `My` defined in `My.idl`, the file `_MyImplBase.java` is generated. You must provide the implementation for `My` and it must inherit from `_MyImplBase`.

  NOTE: ImplBase is deprecated in favor of the POA model, but is provided to allow compatibility with servers written in J2SE 1.3 and prior. We do not recommend creating new servers using this nonstandard model.

- The Delegation Model

  Using the Delegation Model, you implement the IDL interface using two classes:

  - An IDL-generated Tie class that inherits from the compiler-generated skeleton, but delegates all calls to an implementation class.
  - A class that implements the IDL-generated operations interface (such as `HelloOperations`), which defines the IDL function.

  The Delegation model is also known as the *Tie* model, or the Tie Delegation model. It inherits from either the POA or ImplBase compiler-generated skeleton, so the models will be described as POA/Tie or ImplBase/Tie models in this document.

This tutorial presents the POA Inheritance model for server-side implementation. For tutorials using the other server-side implementations, see the following documents:

- *Java IDL: The "Hello World" Example with the POA/Tie Server-Side Model*

  You might want to use the Tie model instead of the typical Inheritance model if your implementation must inherit from some other implementation. Java allows any number of interface inheritance, but there is only one slot for class inheritance. If you use the inheritance model, that slot is used up . By using the Tie Model, that slot is freed up for your own use. The drawback is that it introduces a level of indirection: one extra method call occurs when invoking a method.

- *Java IDL: The "Hello World" Example with the ImplBase Server-Side Model*

  The ImplBase server-side model is an Inheritance Model, as is the POA model. Use the `idlj` compiler with the `-oldImplBase` flag to generate server-side bindings that are compatible with older version of Java IDL (prior to J2SE 1.4).

  Note that using the `-oldImplBase` flag is non-standard: these APIs are being deprecated. You would use this flag ONLY for compatibility with existing servers written in J2SE 1.3 or earlier. In that case, you would need to modify an existing MAKEFILE to add the `-oldImplBase` flag to the `idlj` compiler, otherwise POA-based server-side mappings will be generated.

4 Developing a Client Application

This topic introduces the basics of writing a CORBA client application. Included in this lesson are:

1. Creating HelloClient.java
2. Understanding HelloClient.java
3. Compiling HelloClient.java

Creating HelloClient.java

To create `HelloClient.java`,

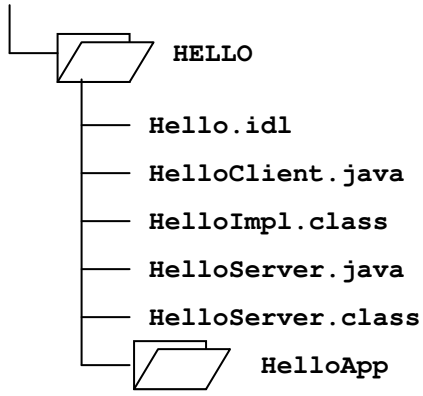
1. Start your text editor and create a file named `HelloClient.java` in your main project directory, `Hello`.
2. Enter the following code for `HelloClient.java` in the text file. The following section, Understanding HelloClient.java, explains each line of code in some detail.

   HelloClient.java

   ```
   // Copyright and License
   ```

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class HelloClient
{
  static Hello helloImpl;

  public static void main(String args[])
    {
      try{
        // create and initialize the ORB
  ORB orb = ORB.init(args, null);

        // get the root naming context
        org.omg.CORBA.Object objRef =
      orb.resolve_initial_references("NameService");
        // Use NamingContextExt instead of NamingContext. This is
        // part of the Interoperable naming Service.
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // resolve the Object Reference in Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

  } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
    e.printStackTrace(System.out);
    }
     }

}
```

3. Save and close `HelloClient.java`.



Understanding HelloClient.java

This section explains each line of `HelloClient.java`, describing what the code does, as well as why it is needed for this application.

Performing Basic Setup

The basic shell of a CORBA client is the same as many Java applications: You import required library packages, declare the application class, define a `main` method, and handle exceptions.

Importing Required Packages

First, we import the packages required for the client class:

```
import HelloApp.*; // the package containing our stubs
import org.omg.CosNaming.*; // HelloClient will use the Naming Service
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*; // All CORBA applications need these classes
```

Declaring the Client Class

The next step is to declare the client class:

```
public class HelloClient
{
  // The main() method goes here.
}
```

Defining a `main()` Method

Every Java application needs a `main()` method. It is declared within the scope of the `HelloClient` class, as follows:

```
  public static void main(String args[])
  {
    // The try-catch block goes here.
  }
```

Handling CORBA System Exceptions

Because all CORBA programs can throw CORBA system exceptions at runtime, all of the `main()` functionality is placed within a try-catch block. CORBA programs throw system exceptions whenever trouble occurs during any of the processes (marshaling, unmarshaling, upcall) involved in invocation.

Our exception handler simply prints the name of the exception and its stack trace to standard output so you can see what kind of thing has gone wrong.

The try-catch block is set up inside `main()`,

```
    try{

      // Add the rest of the HelloClient code here.

    } catch(Exception e) {
        System.out.println("ERROR : " + e);
        e.printStackTrace(System.out);
      }
```

Creating an ORB Object

A CORBA client needs a local ORB object to perform all of its marshaling and IIOP work. Every client instantiates an `org.omg.CORBA.ORB` object and initializes it by passing to the object certain information about itself.

The ORB variable is declared and initialized inside the try-catch block.

```
        ORB orb = ORB.init(args, null);
```

The call to the ORB's `init()` method passes in your application's command line arguments, allowing you to set certain properties at runtime.

Finding the Hello Server

Now that the application has an ORB, it can ask the ORB to locate the actual service it needs, in this case the Hello server. There are a number of ways for a CORBA client to get an initial object reference; our client application will use the COS Naming Service

specified by OMG and provided with Java IDL. See Using Stringified Object References for information on how to get an initial object reference when there is no naming service available.
The two options for Naming Services shipped with J2SE v.1.4 are `orbd`, which is a daemon process containing a Bootstrap Service, a Transient Naming Service, a Persistent Naming Service, and a Server Manager, and `tnameserv`, a transient naming service. This example uses `orbd`.

Obtaining the Initial Naming Context

The first step in using the naming service is to get the initial naming context. In the try-catch block, below your ORB initialization, you call `orb.resolve_initial_references()` to get an object reference to the name server:

```
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
```

The string "NameService" is defined for all CORBA ORBs. When you pass in that string, the ORB returns the initial naming context, an object reference to the name service. The string "NameService" indicates:

- The persistent naming service will be used when using ORBD as the naming service.
- The transient naming service will be used when using `tnameserv` as the naming service.

The string "TNameService" indicates that the transient naming service will be used when ORBD is the naming service. In this example, we are using the persistent naming service that is a part of `orbd`.

Narrowing the Object Reference

As with all CORBA object references, `objRef` is a generic CORBA object. To use it as a `NamingContextExt` object, you must narrow it to its proper type.

```
NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

Here we see the use of an `idlj`-generated helper class, similar in function to `HelloHelper`. The `ncRef` object is now an `org.omg.CosNaming.NamingContextExt` and you can use it to access the naming service and find other services. You will do that in the next step.
The `NamingContextExt` object is new to J2SE v.1.4, and is part of the Interoperable Naming Service.

Resolve the Object Reference in Naming

To publish a reference in the Naming Service to the `Hello` object implementing the `Hello` interface, you first need an identifying string for the `Hello` object.

```
String name = "Hello";
```

Finally, we pass `name` to the naming service's `resolve_str()` method to get an object reference to the Hello server and narrow it to a `Hello` object:

```
helloImpl = HelloHelper.narrow(ncRef.resolve-str(name));
System.out.println("Obtained a handle on server object: " + helloImpl);
```

Here you see the `HelloHelper` helper class at work. The `resolve-str()` method returns a generic CORBA object as you saw above when locating the name service itself. Therefore, you immediately narrow it to a `Hello` object, which is the object reference you need to perform the rest of your work. Then, you send a message to the screen confirming that the object reference has been obtained.

Invoking the `sayHello()` Operation

CORBA invocations look like a method call on a local object. The complications of marshaling parameters to the wire, routing them to the server-side ORB, unmarshaling, and placing the upcall to the server method are completely transparent to the client programmer. Because so much is done for you by generated code, invocation is really the easiest part of CORBA programming.
Finally, we print the results of the invocation to standard output and explicitly shutdown the ORB:

```
System.out.println(helloImpl.sayHello());
helloImpl.shutdown();
```

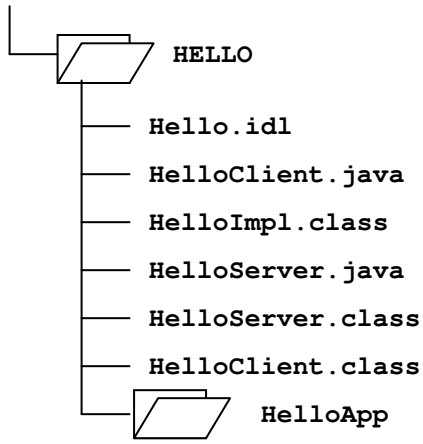
Compiling HelloClient.java

Now we will compile `HelloClient.java` so that we can correct any errors before continuing with this tutorial.
Windows users note that you should substitute backslashes (\) for the slashes (/) in all paths in this document.

To compile `HelloClient.java`,

1. Change to the `Hello` directory.

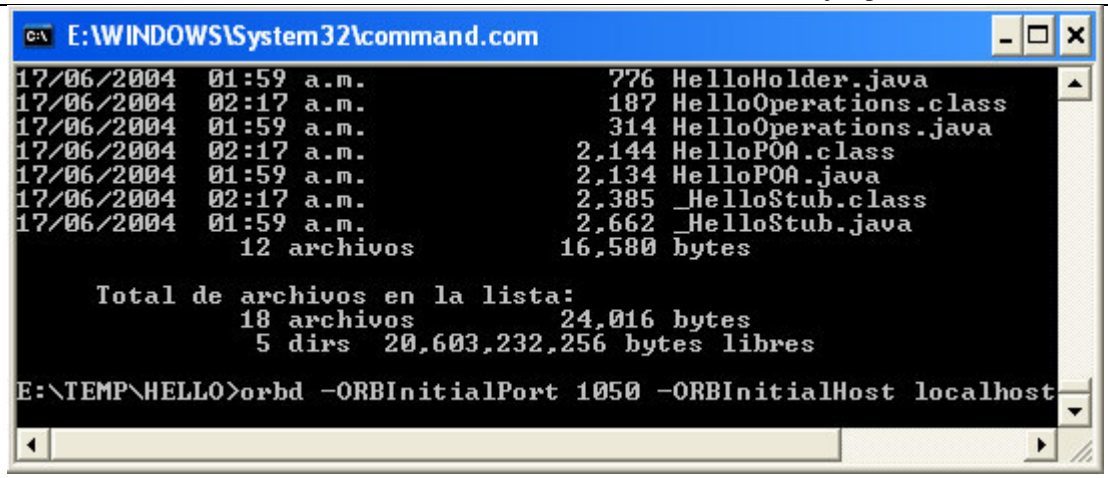
2. Run the Java compiler on `HelloClient.java`:
3. `javac HelloClient.java HelloApp/*.java`
4. Correct any errors in your file and recompile if necessary.
5. The `HelloClient.class` is generated to the `Hello` directory.



5 Running the Hello World Application
This topic walks you through running the server and client program that together make up the "Hello World" application.

Running the Hello World Application
Despite its simple design, the Hello World program lets you learn and experiment with all the tasks required to develop almost any CORBA program that uses static invocation.
This example requires a naming service to make the servant object's operations available to clients. The server needs an object reference to the naming service so that it can publish the references to the objects implementing various interfaces. These object references are used by the clients for invoking methods. The two options for Naming Services shipped with J2SE v.1.4 are `tnameserv`, a transient naming service, and `orbd`, which is a daemon process containing a Bootstrap Service, a Transient Naming Service, a Persistent Naming Service, and a Server Manager. This example uses `orbd`.
When running this example, remember that, when using Solaris software, you must become root to start a process on a port under 1024. For this reason, we recommend that you use a port number greater than or equal to 1024. The `-ORBInitialPort` option is used to override the default port number in this example. The following instructions assume you can use port 1050 for the Java IDL Object Request Broker Daemon, `orbd`. You can substitute a different port if necessary. When running these examples on a Windows machine, subtitute a backslash (\) in path names.
To run this client-server application on your development machine:

1. Start `orbd`.
   To start `orbd` from a UNIX command shell, enter:
   ```
   orbd -ORBInitialPort 1050 -ORBInitialHost localhost&
   ```
   From an MS-DOS system prompt (Windows), enter:
   ```
   start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
   ```



Note that `1050` is the port on which you want the name server to run. `-ORBInitialPort` is a required command-line argument. Note that when using Solaris software, you must become root to start a process on a port under 1024. For this reason, we recommend that you use a port number greater than or equal to 1024.

Note that `-ORBInitialHost` is also a required command-line argument. For this example, since both client and server on running on the development machine, we have set the host to `localhost`. When developing on more than one machine, you will replace this with the name of the host. For an example of how to run this program on two machines, see The Hello World Example on Two Machines.

2. Start the Hello server.

   To start the Hello server from a UNIX command shell, enter:

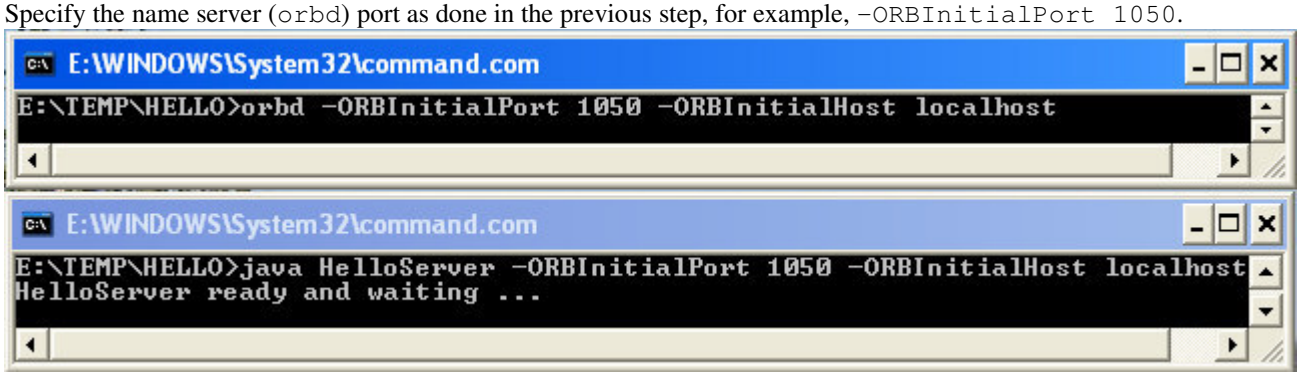
   ```
   java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost&
   ```

   From an MS-DOS system prompt (Windows), enter:

   ```
   start java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost
   ```

   For this example, you can omit `-ORBInitialHost localhost` since the name server is running on the same host as the Hello server. If the name server is running on a different host, use `-ORBInitialHost` *nameserverhost* to specify the host on which the IDL name server is running.

   Specify the name server (`orbd`) port as done in the previous step, for example, `-ORBInitialPort 1050`.



3. Run the client application:

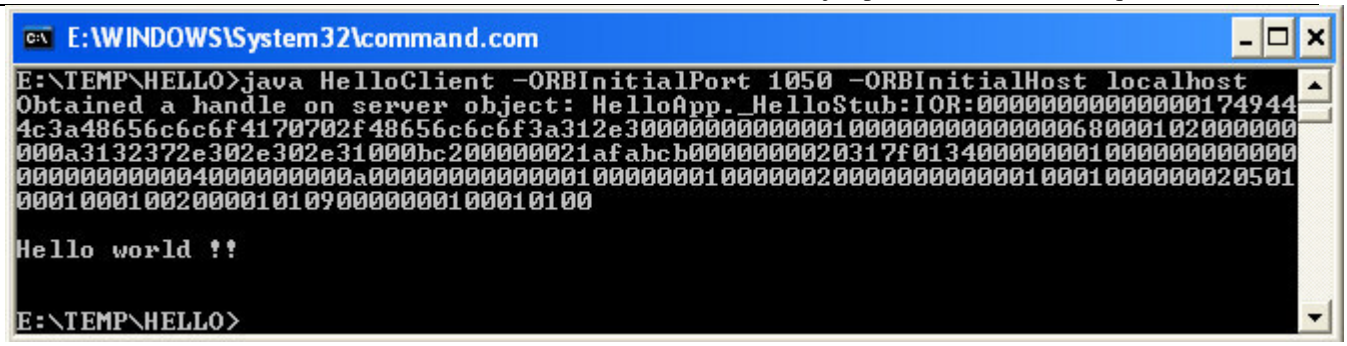
4. `java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost`

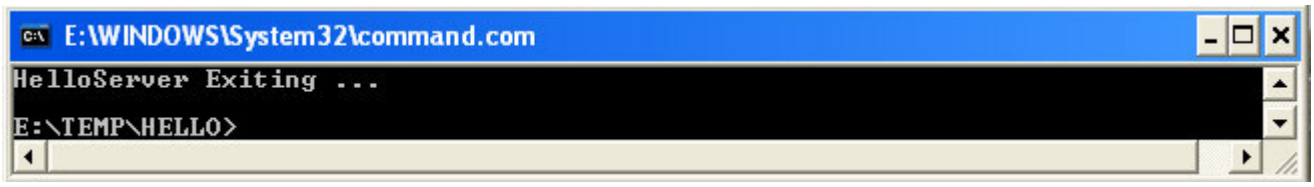
   For this example, you can omit `-ORBInitialHost localhost` since the name server is running on the same host as the Hello client. If the name server is running on a different host, use `-ORBInitialHost` *nameserverhost* to specify the host on which the IDL name server is running.

   Specify the name server (`orbd`) port as done in the previous step, for example, `-ORBInitialPort 1050`.

5. The client prints the string from the server to the command line:

   ```
   Hello world!!
   ```





The name server, like many CORBA servers, runs until you explicitly stop it. To avoid having many servers running, kill the name server process after the client application returns successfully. To do this from a DOS prompt, select the window that is running the server and enter `Ctrl+C` to shut it down. To do this from a Unix shell, find the process, and kill it.

Anexo 11. Ejemplo de desarrollo en Arquitectura Java RMI

Step 1. Defining the Remote Interface

To create an RMI application, the first step is defining of a remote interface between the client and server objects.

```
/* SampleServer.java */
import java.rmi.*;

public interface SampleServer extends Remote
{
  public int sum(int a,int b) throws RemoteException;
}
```

Step 2. Develop the remote object by implement the remote interface

- The server is a simple unicast remote server.
- Create server by extending `java.rmi.server.UnicastRemoteObject`.
- The server uses the `RMISecurityManager` to protect its resources while engaging in remote communication.

```
/* SampleServerImpl.java */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class SampleServerImpl extends UnicastRemoteObject
                             implements SampleServer
{
  SampleServerImpl() throws RemoteException
  {
     super();
  }
```

- The server must bind its name to the registry, the client will look up the server name.
- Use `java.rmi.Naming` class to bind the server name to registry. In this example the name call "SAMPLE-SERVER".
- In the main method of your server object, the RMI security manager is created and installed.

```
  /* SampleServerImpl.java */
  public static void main(String args[])
  {
      try
      {
        System.setSecurityManager(new RMISecurityManager());  //set the security manager

        //create a local instance of the object
        SampleServerImpl Server = new SampleServerImpl();

        //put the local instance in the registry
        Naming.rebind("SAMPLE-SERVER" , Server);

        System.out.println("Server waiting.....");
      }
```

```
      catch (java.net.MalformedURLException me)       {
        System.out.println("Malformed URL: " + me.toString());   }
      catch (RemoteException re)  {
         System.out.println("Remote exception: " + re.toString());  }
  }
```

- Implement the remote mehtods

```
  /* SampleServerImpl.java */
  public int sum(int a,int b) throws RemoteException
  {
     return a + b;
  }
}
```

Step 3. Develop the client program

- In order for the client object to invoke methods on the server, it must first look up the name of server in the registry. You use the `java.rmi.Naming` class to lookup the server name.
- The server name is specified as URL in the from (rmi://host:port/name)
- Default RMI port is 1099.
- The name specified in the URL must exactly match the name that the server has bound to the registry. In this example, the name is "SAMPLE-SERVER"
- The remote method invocation is programmed using the remote interface name (`remoteObject`) as prefix and the remote method name (`sum`) as suffix.

```
import java.rmi.*;
import java.rmi.server.*;

public class SampleClient
{
   public static void main(String[]  args)
   {
      // set the security manager for the client
      System.setSecurityManager(new RMISecurityManager());

      //get the remote object from the registry
      try
        {
          System.out.println("Security Manager loaded");
          String url = "//localhost/SAMPLE-SERVER";

          SampleServer remoteObject = (SampleServer)Naming.lookup(url);
          System.out.println("Got remote object");

          System.out.println(" 1 + 2 = " + remoteObject.sum(1,2) );
        }
        catch (RemoteException exc) {
          System.out.println("Error in lookup: " + exc.toString()); }
        catch (java.net.MalformedURLException exc) {
          System.out.println("Malformed URL: " + exc.toString());   }
        catch (java.rmi.NotBoundException exc)  {
          System.out.println("NotBound: " + exc.toString());
        }

   }
}
```

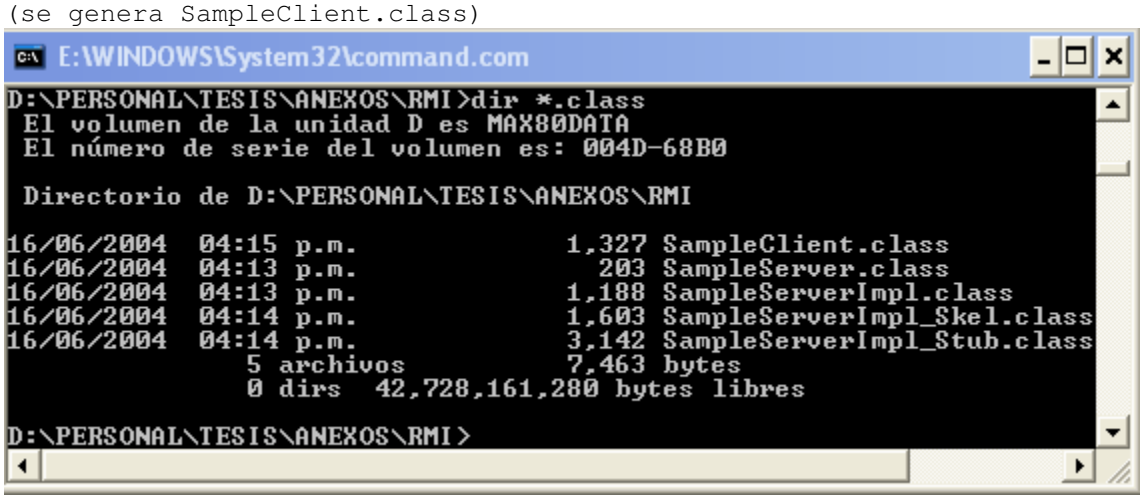
Step 4 & 5. Compile the Java source files & Generate the client stubs and server skeletons

- Assume the program compile and executing at elpis on ~/rmi
- Once the interface is completed, you need to generate stubs and skeleton code. The RMI system provides an RMI compiler (`rmic`) that takes your generated interface class and procedures stub code on its self.

```
elpis:~/rmi> set CLASSPATH=”~/rmi”
elpis:~/rmi> javac SampleServer.java
(se genera SampleServer.class)
elpis:~/rmi> javac SampleServerImpl.java
(se genera SampleServer.class)
elpis:~/rmi> rmic SampleServerImpl
```

(se genera SampleServerImpl_Skel.class y SampleServerImpl_Stub.class)

```
elpis:~/rmi> javac SampleClient.java
(se genera SampleClient.class)
```

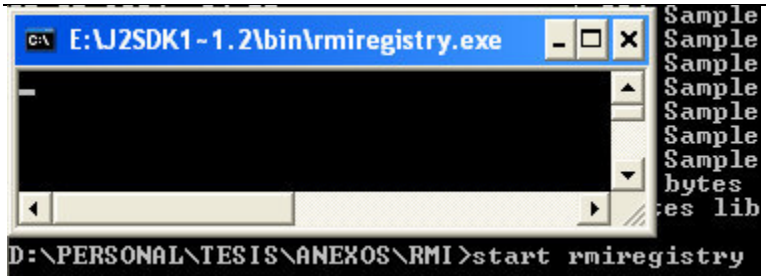


6. Start the RMI registry

- The RMI applications need install to Registry. And the Registry must start manual by call `rmiregisty`.
- The `rmiregistry` us uses port 1099 by default. You can also bind rmiregistry to a different port by indicating the new port number as : `rmiregistry <new port>`

```
elpis:~/rmi> rmiregistry
```

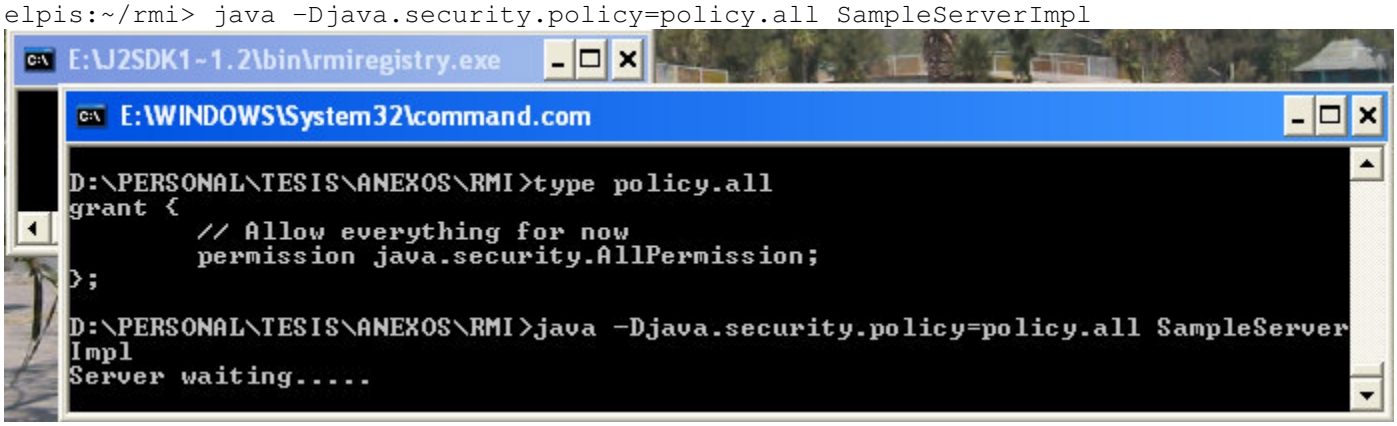
remark: *On Windows, you have to type in from the command line:*
`> start rmiregistry`



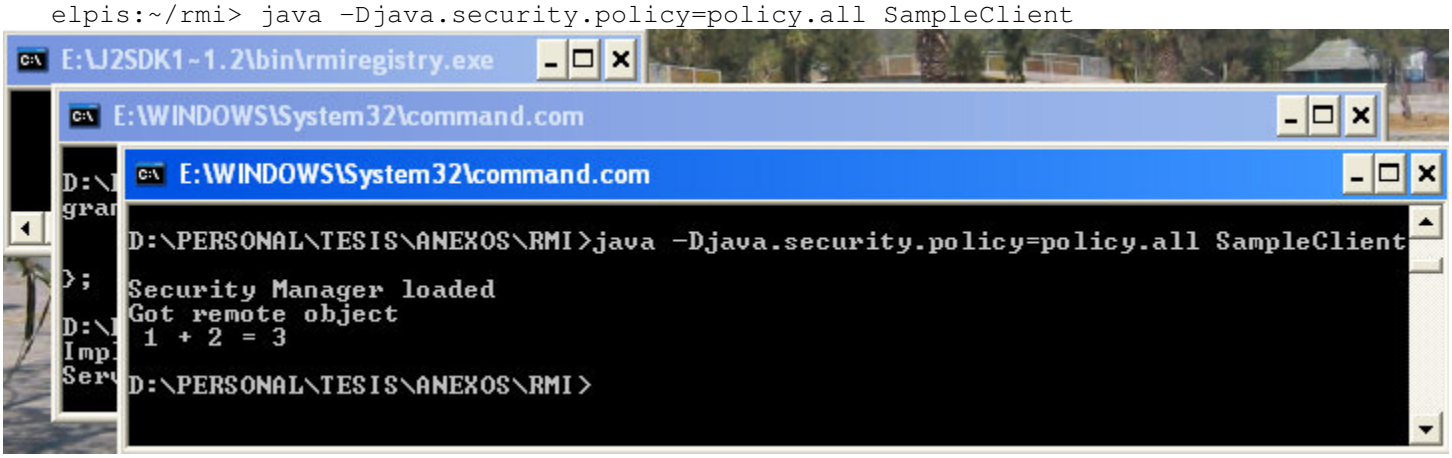
Step 7 & 8. Start the remote server objects & Run the client

- Once the Registry is started, the server can be started and will be able to store itself in the Registry.
- Because of the grained security model in Java 2.0, you must setup a security policy for RMI by set `java.security.policy` to the file `policy.all`

```
elpis:~/rmi> java -Djava.security.policy=policy.all SampleServerImpl
```




```
elpis:~/rmi> java -Djava.security.policy=policy.all SampleClient
```



remark: Java 2 Policy Files

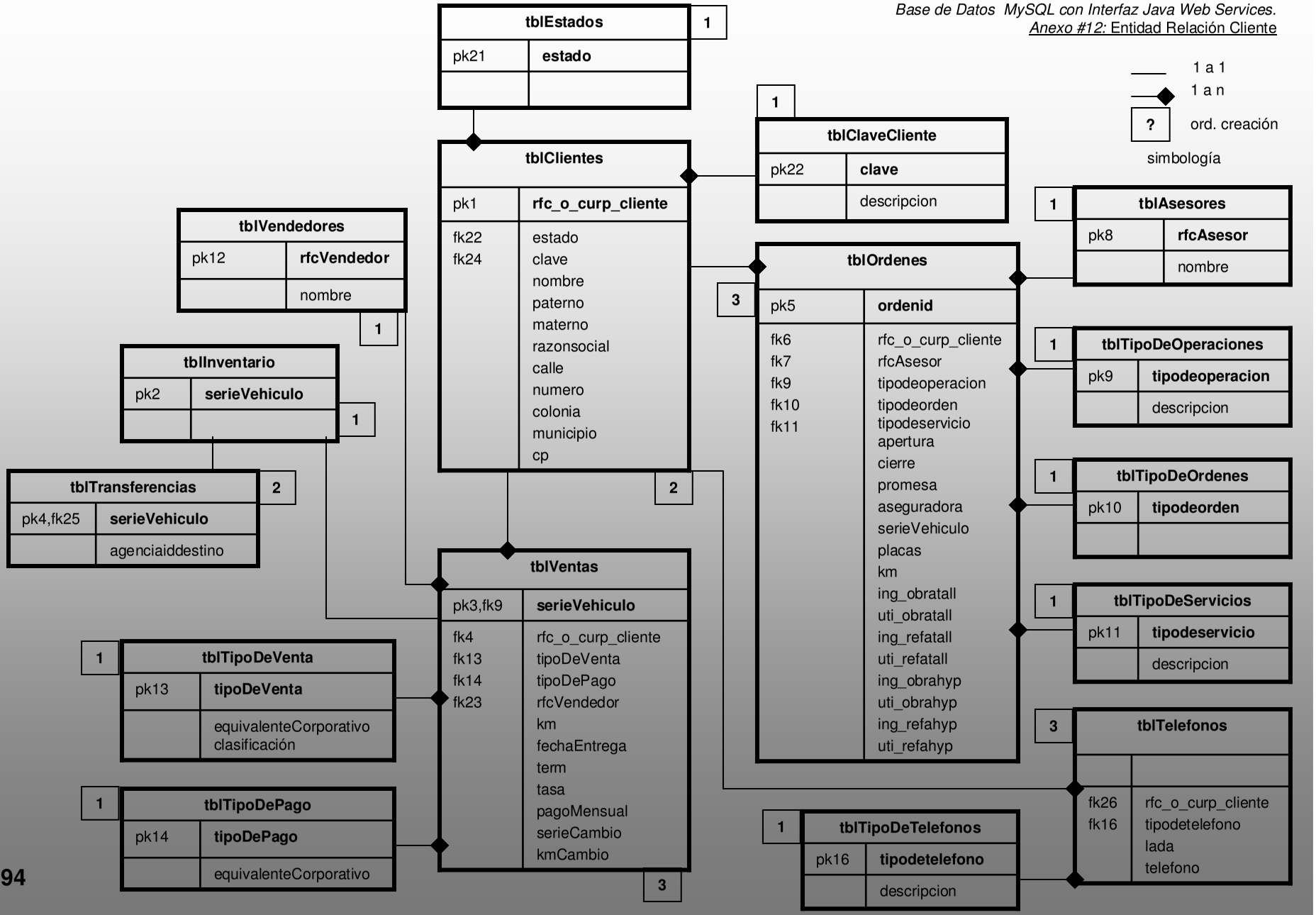
In Java 2, the java application must first obtain information regarding its privileges. It can obtain the security policy through a policy file. In above example, we allow Java code to have all permissions, the contains of the policy file `policy.all` *is:*

```
grant {
      permission java.security.AllPermission;
};
```

Now, we given an example for assigning resource permissions:

```
grant {
      permission java.io.filePermission "/tmp/*", "read", "write";
      permission java.net.SocketPermission "somehost.somedomain.com:999","connect";
      permission java.net.SocketPermission "*:1024-65535","connect,request";
      permission java.net.SocketPermission "*:80","connect";
};
```

1. *allow the Java code to read/write any files only under the /tmp directory, includes any subdirectories*
2. *allow all java classes to establish a network connection with the host "somehost.somedomain.com" on port 999*
3. *allows classes to connection to or accept connections on unprivileged ports greater than 1024 , on any host*
4. *allows all classes to connect to the HTTP port 80 on any host.*


1 a 1
1 a n
?
ord. creación
simbología
tblEstados
pk21 estado
tblClaveCliente
pk22 clave
descripcion
tblClientes
pk1 rfc_o_curp_cliente
fk22 estado
fk24 clave
nombre
paterno
materno
razonsocial
calle
numero
colonia
municipio
cp
tblAsesores
pk8 rfcAsesor
nombre
tblVendedores
pk12 rfcVendedor
nombre
tblOrdenes
pk5 ordenid
fk6 rfc_o_curp_cliente
fk7 rfcAsesor
fk9 tipodeoperacion
fk10 tipodeorden
fk11 tipodeservicio
apertura
cierre
promesa
aseguradora
serieVehiculo
placas
km
ing_obratall
uti_obratall
ing_refatall
uti_refatall
ing_obrahyp
uti_obrahyp
ing_refahyp
uti_refahyp
tblTipoDeOperaciones
pk9 tipodeoperacion
descripcion
tblInventario
pk2 serieVehiculo
tblTipoDeOrdenes
pk10 tipodeorden
tblTransferencias
pk4,fk25 serieVehiculo
agenciaiddestino
tblVentas
pk3,fk9 serieVehiculo
fk4 rfc_o_curp_cliente
fk13 tipoDeVenta
fk14 tipoDePago
fk23 rfcVendedor
km
fechaEntrega
term
tasa
pagoMensual
serieCambio
kmCambio
tblTipoDeServicios
pk11 tipodeservicio
descripcion
tblTipoDeVenta
pk13 tipoDeVenta
equivalenteCorporativo
clasificación
tblTelefonos
fk26 rfc_o_curp_cliente
fk16 tipodetelefono
lada
telefono
tblTipoDePago
pk14 tipoDePago
equivalenteCorporativo
tblTipoDeTelefonos
pk16 tipodetelefono
descripcion

Anexo 13. Creación de base de datos del cliente

Índice de Programas

Regenera

```
Sub Regenera()
Dim strSQL As String
Dim db As DAO.Database, qdf As DAO.QueryDef
Set db = Application.DBEngine.Workspaces(0).Databases(CurrentProject.Path & "\" & CurrentProject.Name)
db.TableDefs.Refresh
db.Relations.Refresh
db.QueryDefs.Refresh
For i = db.Relations.Count - 1 To 0 Step -1
    db.Relations.Delete db.Relations(i).Name
Next i

For i = db.TableDefs.Count - 1 To 0 Step -1
    If db.TableDefs(i).Name = "tblVendedores" Or db.TableDefs(i).Name = "tblInventario" Or _
    db.TableDefs(i).Name = "tblTransferencias" Or db.TableDefs(i).Name = "tblTipoDeVenta" Or _
    db.TableDefs(i).Name = "tblTipoDePago" Or db.TableDefs(i).Name = "tblEstados" Or _
    db.TableDefs(i).Name = "tblClientes" Or db.TableDefs(i).Name = "tblVentas" Or _
    db.TableDefs(i).Name = "tblClaveCliente" Or db.TableDefs(i).Name = "tblOrdenes" Or _
    db.TableDefs(i).Name = "tblTipoDeTelefonos" Or db.TableDefs(i).Name = "tblAsesores" Or _
    db.TableDefs(i).Name = "tblTipoDeOperaciones" Or db.TableDefs(i).Name = "tblTipoDeOrdenes" Or _
    db.TableDefs(i).Name = "tblTipoDeServicios" Or db.TableDefs(i).Name = "tblTelefonos" Or _
    db.TableDefs(i).Name = "tblParametros" Then
        db.TableDefs.Delete db.TableDefs(i).Name
    End If
Next i

strSQL = "CREATE TABLE tblParametros (agenciaid TEXT(20)," + _
"password TEXT(20)," + _
"endpointurl TEXT(200)," + _
"CONSTRAINT PKAgenciaID PRIMARY KEY (agenciaid))"
db.Execute strSQL

' orden 1
strSQL = "CREATE TABLE tblEstados (estado TEXT(20) NOT NULL, " + _
"CONSTRAINT PK21 PRIMARY KEY (estado))"
db.Execute strSQL

strSQL = "CREATE TABLE tblClaveCliente (clave TEXT(20) NOT NULL, " + _
"equivalenteCorporativo text(10), " + _
"descripcion text(50), " + _
"CONSTRAINT PK22 PRIMARY KEY (clave))"
db.Execute strSQL

strSQL = "CREATE TABLE tblAsesores (rfcAsesor TEXT(20) NOT NULL, " + _
"nombre TEXT(50) NOT NULL, " + _
"mAltaAsesor memo, dtAltaAsesor datetime," + _
"mBajaAsesor memo, dtBajaAsesor datetime," + _
"CONSTRAINT PK8 PRIMARY KEY (rfcAsesor))"
db.Execute strSQL

strSQL = "CREATE TABLE tblVendedores (rfcVendedor TEXT(20) NOT NULL, " + _
"nombre TEXT(50) NOT NULL, " + _
"mAltaVendedor memo, dtAltaVendedor DATETIME," + _
"mBajaVendedor memo, dtBajaVendedor DATETIME," + _
"CONSTRAINT PK12 PRIMARY KEY (rfcVendedor))"
db.Execute strSQL


strSQL = "CREATE TABLE tblTipoDeOperaciones (tipodeoperacion TEXT(20) NOT NULL, " + _
"descripcion TEXT(50), " + _
"CONSTRAINT PK9 PRIMARY KEY (tipodeoperacion))"
db.Execute strSQL

strSQL = "CREATE TABLE tblTipoDeOrdenes (tipodeorden TEXT(20) NOT NULL, " + _
"CONSTRAINT PK10 PRIMARY KEY (tipodeorden))"
db.Execute strSQL

strSQL = "CREATE TABLE tblTipoDeServicios (tipodeservicio TEXT(20) NOT NULL, " + _
"CONSTRAINT PK11 PRIMARY KEY (tipodeservicio))"
db.Execute strSQL

strSQL = "CREATE TABLE tblTipoDeTelefonos (tipodetelefono TEXT(20) NOT NULL, " + _
"descripcion TEXT(50), " + _
"CONSTRAINT PK16 PRIMARY KEY (tipodetelefono))"
db.Execute strSQL

strSQL = "CREATE TABLE tblTipoDeVenta (tipodeventa TEXT(20) NOT NULL, " + _
"equivalentecorporativo TEXT(10), " + _
```

```
"clasificacion TEXT(100), " + _
"CONSTRAINT PK13 PRIMARY KEY (tipodeventa))"
db.Execute strSQL

strSQL = "CREATE TABLE tblTipoDePago (tipodepago TEXT(20) NOT NULL, " + _
"equivalentecorporativo TEXT(10), " + _
"CONSTRAINT PK14 PRIMARY KEY (tipodepago))"
db.Execute strSQL

strSQL = "CREATE TABLE tblInventario (serieVehiculo TEXT(20) NOT NULL, " + _
"mAltaUnidad memo, dtAltaUnidad DATETIME," + _
"mBajaUnidad memo, dtBajaUnidad DATETIME," + _
"mAltaInventario memo, dtAltaInventario DATETIME," + _
"CONSTRAINT PK2 PRIMARY KEY (serieVehiculo))"
db.Execute strSQL

' orden 2
strSQL = "CREATE TABLE tblTransferencias (serieVehiculo TEXT(20) NOT NULL, " + _
"agenciaiddestino text(20) NOT NULL, " + _
"mTransferencia MEMO, dtTransferencia DATETIME, " + _
"CONSTRAINT PK4 PRIMARY KEY (serieVehiculo), " + _
"CONSTRAINT FK8 FOREIGN KEY (serieVehiculo) REFERENCES tblInventario) "
db.Execute strSQL

strSQL = "CREATE TABLE tblClientes (rfc_o_curp_cliente text(20) NOT NULL, " + _
"estado TEXT(20) NOT NULL, clave TEXT(20) NOT NULL , nombre TEXT(100) NOT NULL , " + _
"paterno TEXT(100), materno TEXT(100) NOT NULL, razonSocial TEXT(100) NOT NULL, " + _
"calle TEXT(50), numero TEXT(50) NOT NULL, colonia TEXT(50) NOT NULL, " + _
"municipio TEXT(50), cp TEXT(10) NOT NULL, " + _
"mAltaCliente MEMO, dtAltaCliente DATETIME, " + _
"mBajaCliente MEMO, dtBajaCliente DATETIME, " + _
"CONSTRAINT PK1 PRIMARY KEY (rfc_o_curp_cliente), " + _
"CONSTRAINT FK24 FOREIGN KEY (clave) REFERENCES tblClaveCliente, " + _
"CONSTRAINT FK22 FOREIGN KEY (estado) REFERENCES tblEstados) "
'InputBox "mensaje", "titulo", strSQL
db.Execute strSQL

' orden 3
strSQL = "CREATE TABLE tblTelefonos (rfc_o_curp_cliente text(20) NOT NULL, " + _
"tipodetelefono TEXT(20) NOT NULL, " + _
"lada TEXT(20), telefono TEXT(20) NOT NULL, " + _
"CONSTRAINT FK26 FOREIGN KEY (rfc_o_curp_cliente) REFERENCES tblClientes, " + _
"CONSTRAINT FK16 FOREIGN KEY (tipodetelefono) REFERENCES tblTipoDeTelefonos) "
'InputBox "mensaje", "titulo", strSQL
db.Execute strSQL

strSQL = "CREATE TABLE tblOrdenes (ordenid TEXT(20) NOT NULL, " + _
"rfc_o_curp_cliente TEXT(20) NOT NULL , rfcAsesor TEXT(20) NOT NULL, " + _
"tipodeoperacion TEXT(20) NOT NULL, tipodeorden TEXT(20) NOT NULL, " + _
"tipodeservicio TEXT(20) NOT NULL, " + _
"apertura DATETIME, cierre DATETIME, promesa DATETIME, " + _
"aseguradora TEXT(100) NOT NULL,  " + _
"serieVehiculo TEXT(20), placas TEXT(20) , km INTEGER , " + _
"ing_obratall MONEY , uti_obratall MONEY, ing_refatall MONEY," + _
"uti_refatall MONEY, ing_obrahyp MONEY, uti_obrahyp MONEY, " + _
"ing_refahyp  MONEY, uti_refahyp MONEY, " + _
"CONSTRAINT PK5 PRIMARY KEY (ordenid), " + _
"CONSTRAINT FK6 FOREIGN KEY (rfc_o_curp_cliente) REFERENCES tblClientes, " + _
"CONSTRAINT FK7 FOREIGN KEY (rfcAsesor) REFERENCES tblAsesores,  " + _
"CONSTRAINT FK9 FOREIGN KEY (tipodeoperacion) REFERENCES tblTipoDeOperaciones, " + _
"CONSTRAINT FK10 FOREIGN KEY (tipodeorden) REFERENCES tblTipoDeOrdenes, " + _
"CONSTRAINT FK11 FOREIGN KEY (tipodeservicio) REFERENCES tblTipoDeServicios) "
'InputBox "mensaje", "titulo", strSQL
db.Execute strSQL

strSQL = "ALTER TABLE tblOrdenes ADD COLUMN mAltaOrden MEMO"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN dtAltaOrden DATETIME"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN mImportesOrden MEMO"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN dtImportesOrden DATETIME"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN mCierraOrden MEMO"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN dtCierraOrden DATETIME"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN mCancelaOrden MEMO"
db.Execute strSQL
strSQL = "ALTER TABLE tblOrdenes ADD COLUMN dtCancelaOrden DATETIME"
```

```
db.Execute strSQL


strSQL = "CREATE TABLE tblVentas (serieVehiculo TEXT(20) NOT NULL, " + _
"rfc_o_curp_cliente TEXT(20) NOT NULL, " + _
"tipoDeVenta TEXT(20) NOT NULL, " + _
"tipoDePago TEXT(20) NOT NULL, " + _
"rfcVendedor TEXT(20) NOT NULL, " + _
"km INTEGER, fechaEntrega DATETIME, " + _
"term INTEGER, " + _
"tasa REAL, " + _
"pagoMensual MONEY, " + _
"serieCambio TEXT(20), " + _
"kmCambio INTEGER, " + _
"mReportaVenta MEMO, dtReportaVenta DATETIME," + _
"mCancelaVenta MEMO, dtCancelaVenta DATETIME," + _
"CONSTRAINT PK3 PRIMARY KEY (serieVehiculo), " + _
"CONSTRAINT FK4 FOREIGN KEY (rfc_o_curp_cliente) REFERENCES tblClientes, " + _
"CONSTRAINT FK13 FOREIGN KEY (tipoDeVenta) REFERENCES tblTipoDeVenta, " + _
"CONSTRAINT FK14 FOREIGN KEY (tipoDePago) REFERENCES tblTipoDePago, " + _
"CONSTRAINT FK236 FOREIGN KEY (rfcVendedor) REFERENCES tblVendedores, " + _
"CONSTRAINT FK109 FOREIGN KEY (serieVehiculo) REFERENCES tblInventario)"
db.Execute strSQL


db.Close
AlimentaCatalogos
End Sub
```

AlimentaCatalogos

```
Sub AlimentaCatalogos()
Dim db As DAO.Database, qdf As DAO.QueryDef
Dim parameterArray As Variant, strSQL As String
Set db = Application.DBEngine.Workspaces(0).Databases(CurrentProject.Path & "\" & CurrentProject.Name)
db.TableDefs.Refresh
db.Relations.Refresh
db.QueryDefs.Refresh

ReDim parameterArray(0 To 0, 1 To 3)
parameterArray(0, 1) = "M1188"
parameterArray(0, 2) = "moreno"
parameterArray(0, 3) = "http://localhost:8080/produccion/servlet/ServicioHTTP"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblParametros values ('" + _
    parameterArray(i, 1) + "','" + _
    parameterArray(i, 2) + "','" + _
    parameterArray(i, 3) + "')"
    db.Execute strSQL
Next i

parameterArray = Array("AGS", "BC", "BCS", "CAMP", "CHIS", "CHIH", "COAH", _
"COL", "DF", "DGO", "MEX", "GTO", "GRO", "HGO", "JAL", "MICH", "MOR", "NAY", _
"NL", "OAX", "PUE", "QRO", "Q.ROO", "SLP", "SIN", "SON", "TAB", "TAMPS", "TLAX", _
"VER", "YUC", "ZAC")
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblEstados values ('" + parameterArray(i) + "')"
    db.Execute strSQL
Next i

ReDim parameterArray(0 To 2, 1 To 2)
parameterArray(0, 1) = "mant"
parameterArray(0, 2) = "mantenimiento"
parameterArray(1, 1) = "repma"
parameterArray(1, 2) = "reparación mayor"
parameterArray(2, 1) = "hyp"
parameterArray(2, 2) = "hojalatería y pintura"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDeOperaciones values ('" + parameterArray(i, 1) + "','" + _
    parameterArray(i, 2) + "')"
    db.Execute strSQL
Next i

parameterArray = Array("publico", "interno", "garantia")
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDeOrdenes values ('" + parameterArray(i) + "')"
    db.Execute strSQL
Next i
```

```
parameterArray = Array("paquete", "express")
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDeServicios values ('" + parameterArray(i) + "')"
    db.Execute strSQL
Next i

ReDim parameterArray(0 To 8, 1 To 3)
parameterArray(0, 1) = "vmenudeo"
parameterArray(0, 2) = "0"
parameterArray(0, 3) = "menudeo - retail"
parameterArray(1, 1) = "vempleado"
parameterArray(1, 2) = "A"
parameterArray(1, 3) = "menudeo - empleados de contado"
parameterArray(2, 1) = "vcomgere"
parameterArray(2, 2) = "X"
parameterArray(2, 3) = "menudeo - compromisos gerenciales"
parameterArray(3, 1) = "vnarrend"
parameterArray(3, 2) = "6"
parameterArray(3, 3) = "vmenudeo - arrendamiento individual a través de arrendadora independiente"
parameterArray(4, 1) = "varrend"
parameterArray(4, 2) = "2"
parameterArray(4, 3) = "menudeo - redcarpet multiopcion"
parameterArray(5, 1) = "flotmenor"
parameterArray(5, 2) = "E"
parameterArray(5, 3) = "flotilla - compromiso de la gerencia"
parameterArray(6, 1) = "flotmayor"
parameterArray(6, 2) = "7"
parameterArray(6, 3) = "flotilla - flotilleros comerciales"
parameterArray(7, 1) = "vtagobfed"
parameterArray(7, 2) = "S"
parameterArray(7, 3) = "flotilla - gobierno federal"
parameterArray(8, 1) = "vtagobest"
parameterArray(8, 2) = "3"
parameterArray(8, 3) = "flotilla - gobierno estatal"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDeVenta values ('" + parameterArray(i, 1) + _
    "','" + parameterArray(i, 2) + "','" + parameterArray(i, 3) + "')"
    db.Execute strSQL
Next i

ReDim parameterArray(0 To 2, 1 To 2)
parameterArray(0, 1) = "arrendamiento"
parameterArray(0, 2) = "L"
parameterArray(1, 1) = "financiamiento"
parameterArray(1, 2) = "F"
parameterArray(2, 1) = "contado"
parameterArray(2, 2) = "C"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDePago values ('" + _
    parameterArray(i, 1) + "','" + parameterArray(i, 2) + "')"
    db.Execute strSQL
Next i

ReDim parameterArray(0 To 5, 1 To 2)
parameterArray(0, 1) = "compradorcasa"
parameterArray(0, 2) = "Telefono de Casa de Comprador"
parameterArray(1, 1) = "compradoroficina"
parameterArray(1, 2) = "Telefono de Oficina de Comprador"
parameterArray(2, 1) = "compradorcel"
parameterArray(2, 2) = "Telefono Celular de Comprador"
parameterArray(3, 1) = "conductorcasa"
parameterArray(3, 2) = "Telefono de Casa de Conductor"
parameterArray(4, 1) = "conductoroficina"
parameterArray(4, 2) = "Telefono de Oficina de Conductor"
parameterArray(5, 1) = "conductorcel"
parameterArray(5, 2) = "Telefono Celular de Conductor"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblTipoDeTelefonos values ('" + _
    parameterArray(i, 1) + "','" + parameterArray(i, 2) + "')"
    db.Execute strSQL
Next i

ReDim parameterArray(0 To 5, 1 To 3)
parameterArray(0, 1) = "negocios"
parameterArray(0, 2) = "0"
parameterArray(0, 3) = "venta de negocios"
parameterArray(1, 1) = "sr"
parameterArray(1, 2) = "1"
parameterArray(1, 3) = "señor"
```

```
parameterArray(2, 1) = "sra"
parameterArray(2, 2) = "2"
parameterArray(2, 3) = "señora"
parameterArray(3, 1) = "srta"
parameterArray(3, 2) = "3"
parameterArray(3, 3) = "señorita"
parameterArray(4, 1) = "extranjero"
parameterArray(4, 2) = "8"
parameterArray(4, 3) = "venta al extranjero"
parameterArray(5, 1) = "moral"
parameterArray(5, 2) = "9"
parameterArray(5, 3) = "venta a persona moral"
For i = 0 To UBound(parameterArray, 1)
    strSQL = "INSERT INTO tblClaveCliente values ('" + _
    parameterArray(i, 1) + "','" + parameterArray(i, 2) + _
    "','" + parameterArray(i, 3) + "')"
    db.Execute strSQL
Next i
db.Close
End Sub
```

Anexo 14. Proxies en el Cliente.

Índice de Programas

Declaraciones de Variables y Tipos de Datos

```
Option Compare Database
Option Explicit
Global END_POINT_URL As String
'Private Const END_POINT_URL = "http://19.192.54.158/cgi-bin/lectura.cgi"
Type DatosCliente
    agenciaid As String
    password As String
    rfc_o_curp_cliente As String
    estado As String
    clave As String
    nombre As String
    paterno As String
    materno As String
    razonsocial As String
    calle As String
    numero As String
    colonia As String
    municipio As String
    cp As String
End Type
```

BajaCliente

```
Function BajaCliente(ByVal agenciaid As String, ByVal password As String, _
ByVal rfc_o_curp_cliente As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/BajaCliente"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "BajaCliente"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "rfc_o_curp_cliente"
                    Serializer.WriteString rfc_o_curp_cliente
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom

    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        BajaCliente = Reader.FaultString.Text
    Else
        BajaCliente = Reader.RpcResult.Text
    End If
End Function
```

Transferencia

```
Function Transferencia(ByVal agenciaid As String, ByVal password As String, _
```

```
ByVal serieVehiculo As String, ByVal agenciaiddestino As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/Transferencia"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "Transferencia"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "serieVehiculo"
                    Serializer.WriteString serieVehiculo
                Serializer.EndElement
                Serializer.StartElement "agenciaiddestino"
                    Serializer.WriteString agenciaiddestino
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        Transferencia = Reader.FaultString.Text
    Else
        Transferencia = Reader.RpcResult.Text
    End If

End Function
```

CancelaVenta

```
Function CancelaVenta(ByVal agenciaid As String, ByVal password As String, _
ByVal serieVehiculo As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/CancelaVenta"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "CancelaVenta"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
```

```
                Serializer.EndElement
                Serializer.StartElement "serieVehiculo"
                    Serializer.WriteString serieVehiculo
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        CancelaVenta = Reader.FaultString.Text
    Else
        CancelaVenta = Reader.RpcResult.Text
    End If

End Function
```

BajaUnidad

```
Function BajaUnidad(ByVal agenciaid As String, ByVal password As String, _
ByVal serieVehiculo As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/BajaUnidad"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "BajaUnidad"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "serieVehiculo"
                    Serializer.WriteString serieVehiculo
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        BajaUnidad = Reader.FaultString.Text
    Else
        BajaUnidad = Reader.RpcResult.Text
    End If

End Function
```

BajaVendedor

```
Function BajaVendedor(ByVal agenciaid As String, ByVal password As String, _
```

```
ByVal rfcVendedor As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/BajaVendedor"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "BajaVendedor"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "rfcVendedor"
                    Serializer.WriteString rfcVendedor
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        BajaVendedor = Reader.FaultString.Text
    Else
        BajaVendedor = Reader.RpcResult.Text
    End If
End Function
```

BajaAsesor

```
Function BajaAsesor(ByVal agenciaid As String, ByVal password As String, _
ByVal rfcasesor As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/BajaAsesor"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "BajaAsesor"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "rfcAsesor"
                    Serializer.WriteString rfcasesor
                Serializer.EndElement
            Serializer.EndElement
```

```
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        BajaAsesor = Reader.FaultString.Text
    Else
        BajaAsesor = Reader.RpcResult.Text
    End If

End Function
```

AltaAsesor

```
Function AltaAsesor(ByVal agenciaid As String, ByVal password As String, _
ByVal rfcasesor As String, ByVal nombre As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/AltaAsesor"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "AltaAsesor"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "rfcAsesor"
                    Serializer.WriteString rfcasesor
                Serializer.EndElement
                Serializer.StartElement "nombre"
                    Serializer.WriteString nombre
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        AltaAsesor = Reader.FaultString.Text
    Else
        AltaAsesor = Reader.RpcResult.Text
    End If
End Function
```

AltaVendedor

```
Function AltaVendedor(ByVal agenciaid As String, ByVal password As String, _
ByVal rfcVendedor As String, ByVal nombre As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
```

```
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/AltaVendedor"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "AltaVendedor"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "rfcVendedor"
                    Serializer.WriteString rfcVendedor
                Serializer.EndElement
                Serializer.StartElement "nombre"
                    Serializer.WriteString nombre
                Serializer.EndElement
            Serializer.EndElement 'fin de addUnit
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        AltaVendedor = Reader.FaultString.Text
    Else
        AltaVendedor = Reader.RpcResult.Text
    End If
End Function
```

CierraOrden

```
Function CierraOrden(ByVal agenciaid As String, ByVal password As String, _
ByVal ordenid As String, ByVal cierre As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/CierraOrden"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "CierraOrden"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "ordenid"
                    Serializer.WriteString ordenid
                Serializer.EndElement
                Serializer.StartElement "cierre"
```

```
                    Serializer.WriteString cierre
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        CierraOrden = Reader.FaultString.Text
    Else
        CierraOrden = Reader.RpcResult.Text
    End If

End Function
```

CancelaOrden

```
Function CancelaOrden(ByVal agenciaid As String, ByVal password As String, _
ByVal ordenid As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/CancelaOrden"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "CancelaOrden"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "ordenid"
                    Serializer.WriteString ordenid
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        CancelaOrden = Reader.FaultString.Text
    Else
        CancelaOrden = Reader.RpcResult.Text
    End If
End Function
```

ImportesOrden

```
Function ImportesOrden(ByVal agenciaid As String, ByVal password As String, _
ByVal ordenid As String, ByVal ing_obratall As String, ByVal uti_obratall As String, _
ByVal ing_refatall As String, ByVal uti_refatall As String, ByVal ing_obrahyp, _
ByVal uti_obrahyp As String, ByVal ing_refahyp As String, ByVal uti_refahyp As String)
```

```
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/ImportesOrden"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "ImportesOrden"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "ordenid"
                    Serializer.WriteString ordenid
                Serializer.EndElement
                Serializer.StartElement "ing_obratall"
                    Serializer.WriteString ing_obratall
                Serializer.EndElement
                Serializer.StartElement "uti_obratall"
                    Serializer.WriteString uti_obratall
                Serializer.EndElement
                Serializer.StartElement "ing_refatall"
                    Serializer.WriteString ing_refatall
                Serializer.EndElement
                Serializer.StartElement "uti_refatall"
                    Serializer.WriteString uti_refatall
                Serializer.EndElement
                Serializer.StartElement "ing_obrahyp"
                    Serializer.WriteString ing_obrahyp
                Serializer.EndElement
                Serializer.StartElement "uti_obrahyp"
                    Serializer.WriteString uti_obrahyp
                Serializer.EndElement
                Serializer.StartElement "ing_refahyp"
                    Serializer.WriteString ing_refahyp
                Serializer.EndElement
                Serializer.StartElement "uti_refahyp"
                    Serializer.WriteString uti_refahyp
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement

    If Not Reader.Fault Is Nothing Then
        ImportesOrden = Reader.FaultString.Text
    Else
        ImportesOrden = Reader.RpcResult.Text
    End If

End Function
```

AltaOrden

```
Function AltaOrden(ByVal agenciaid As String, ByVal password As String, ByVal ordenid As String, _
ByVal rfc_o_curp_cliente As String, ByVal rfcasesor As String, ByVal tipodeoperacion As String, _
ByVal tipodeorden As String, ByVal tipodeservicio As String, ByVal apertura As String, ByVal cierre As String, _
ByVal promesa As String, ByVal aseguradora As String, ByVal serieVehiculo As String, ByVal placas As String, _
ByVal km As String)
    Dim Serializer As SoapSerializer30
```

```
Dim Reader As SoapReader30
Dim ResultElm As IXMLDOMElement
Dim FaultElm As IXMLDOMElement
Dim Connector As ISoapConnector
Dim i As Integer, cadena As String
Set Connector = New HttpConnector30
Connector.Property("EndPointURL") = END_POINT_URL
Connector.Connect

Connector.Property("SoapAction") = "http://www.unam.mx/AltaOrden"
Connector.BeginMessage

Set Serializer = New SoapSerializer30
Serializer.Init Connector.InputStream
Serializer.StartEnvelope
    Serializer.StartBody
        Serializer.StartElement "AltaOrden"
            Serializer.StartElement "agenciaid"
                Serializer.WriteString agenciaid
            Serializer.EndElement
            Serializer.StartElement "password"
                Serializer.WriteString password
            Serializer.EndElement
            Serializer.StartElement "ordenid"
                Serializer.WriteString ordenid
            Serializer.EndElement
            Serializer.StartElement "rfc_o_curp_cliente"
                Serializer.WriteString rfc_o_curp_cliente
            Serializer.EndElement
            Serializer.StartElement "rfcAsesor"
                Serializer.WriteString rfcasesor
            Serializer.EndElement
            Serializer.StartElement "tipodeoperacion"
                Serializer.WriteString tipodeoperacion
            Serializer.EndElement
            Serializer.StartElement "tipodeorden"
                Serializer.WriteString tipodeorden
            Serializer.EndElement
            Serializer.StartElement "tipodeservicio"
                Serializer.WriteString tipodeservicio
            Serializer.EndElement
            Serializer.StartElement "apertura"
                Serializer.WriteString apertura
            Serializer.EndElement
            Serializer.StartElement "cierre"
                Serializer.WriteString cierre
            Serializer.EndElement
            Serializer.StartElement "promesa"
                Serializer.WriteString promesa
            Serializer.EndElement
            Serializer.StartElement "aseguradora"
                Serializer.WriteString aseguradora
            Serializer.EndElement
            Serializer.StartElement "serieVehiculo"
                Serializer.WriteString serieVehiculo
            Serializer.EndElement
            Serializer.StartElement "placas"
                Serializer.WriteString placas
            Serializer.EndElement
            Serializer.StartElement "km"
                Serializer.WriteString km
            Serializer.EndElement
        Serializer.EndElement
    Serializer.EndBody
Serializer.EndEnvelope
Connector.EndMessage

Set Reader = New SoapReader30
Reader.Load Connector.OutputStream
Dim Dom As MSXML2.DOMDocument30
Dim Elm As IXMLDOMElement
Set Dom = Reader.Dom
Set Elm = Dom.documentElement

If Not Reader.Fault Is Nothing Then
    AltaOrden = Reader.FaultString.Text
Else
    AltaOrden = Reader.RpcResult.Text
End If
```

```
End Function
```

ReportaVenta

```
Function ReportaVenta(ByVal agenciaid As String, ByVal password As String, _
ByVal serieVehiculo As String, ByVal rfc_o_curp_cliente As String, _
ByVal tipoVenta As String, ByVal tipoPago As String, _
ByVal rfcVendedor As String, ByVal km As String, ByVal fechaEntrega As String, _
ByVal term As String, ByVal tasa As String, ByVal pagoMensual As String, _
ByVal serieCambio As String, ByVal kmCambio As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/ReportaVenta"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "ReportaVenta"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "serieVehiculo"
                    Serializer.WriteString serieVehiculo
                Serializer.EndElement
                Serializer.StartElement "rfc_o_curp_cliente"
                    Serializer.WriteString rfc_o_curp_cliente
                Serializer.EndElement
                Serializer.StartElement "tipoventa"
                    Serializer.WriteString tipoVenta
                Serializer.EndElement
                Serializer.StartElement "tipopago"
                    Serializer.WriteString tipoPago
                Serializer.EndElement
                Serializer.StartElement "rfcVendedor"
                    Serializer.WriteString rfcVendedor
                Serializer.EndElement
                Serializer.StartElement "km"
                    Serializer.WriteString km
                Serializer.EndElement
                Serializer.StartElement "fechaEntrega"
                    Serializer.WriteString fechaEntrega
                Serializer.EndElement
                Serializer.StartElement "term"
                    Serializer.WriteString term
                Serializer.EndElement
                Serializer.StartElement "tasa"
                    Serializer.WriteString tasa
                Serializer.EndElement
                Serializer.StartElement "pagoMensual"
                    Serializer.WriteString pagoMensual
                Serializer.EndElement
                Serializer.StartElement "serieCambio"
                    Serializer.WriteString serieCambio
                Serializer.EndElement
                Serializer.StartElement "kmCambio"
                    Serializer.WriteString kmCambio
                Serializer.EndElement
            Serializer.EndElement 'fin de addUnit
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream

    If Not Reader.Fault Is Nothing Then
```

```
        ' Leer solo el mensaje de RPC
        ReportaVenta = Reader.FaultString.Text
    Else
        ' Leer solo el mensaje de RPC
        ReportaVenta = Reader.RpcResult.Text
    End If

    ' Leer todo el xml
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement
    ReportaVenta = Dom.XML
End Function
```

ListaAsesores

```
Function ListaAsesores(ByVal agenciaid As String, ByVal password As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/ListaAsesores"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "ListaAsesores"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement
    If Not Reader.Fault Is Nothing Then
        ListaAsesores = Reader.FaultString.Text
    Else
        ListaAsesores = Dom.XML
    End If

End Function
```

ListaVendedores

```
Function ListaVendedores(ByVal agenciaid As String, ByVal password As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/ListaVendedores"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
```

```
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "ListaVendedores"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement
    If Not Reader.Fault Is Nothing Then
        ListaVendedores = Reader.FaultString.Text
    Else
        ListaVendedores = Dom.XML
    End If

End Function
```

ListaClientes

```
Function ListaClientes(ByVal agenciaid As String, ByVal password As String)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/ListaClientes"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "ListaClientes"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement
    If Not Reader.Fault Is Nothing Then
        ListaClientes = Reader.FaultString.Text
    Else
        ListaClientes = Dom.XML
    End If
End Function
```

AltaUnidad

```
Function AltaUnidad(ByVal agenciaid As String, ByVal password As String, _
ByVal serieVehiculo As String)
```

```
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/AltaUnidad"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "AltaUnidad"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.StartElement "serieVehiculo"
                    Serializer.WriteString serieVehiculo
                Serializer.EndElement
            Serializer.EndElement 'fin de addUnit
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream
    Dim Dom As MSXML2.DOMDocument30
    Dim Elm As IXMLDOMElement
    Set Dom = Reader.Dom
    Set Elm = Dom.documentElement
    'MsgBox Dom.XML
    'MsgBox Elm.XML
    If Not Reader.Fault Is Nothing Then
        AltaUnidad = Reader.FaultString.Text
    Else
        AltaUnidad = Reader.RpcResult.Text
    End If
End Function
```

AltaCliente

```
Public Function AltaCliente(cliente As DatosCliente, ByRef arrayTelefonos)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/AltaCliente"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "AltaCliente"
                Serializer.StartElement "agenciaid"
                    Serializer.WriteString cliente.agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString cliente.password
                Serializer.EndElement
                Serializer.StartElement "rfc_o_curp_cliente"
                    Serializer.WriteString cliente.rfc_o_curp_cliente
                Serializer.EndElement
                Serializer.StartElement "estado"
                    Serializer.WriteString cliente.estado
```

```
                Serializer.EndElement
                Serializer.StartElement "clave"
                    Serializer.WriteString cliente.clave
                Serializer.EndElement
                Serializer.StartElement "nombre"
                    Serializer.WriteString cliente.nombre
                Serializer.EndElement
                Serializer.StartElement "paterno"
                    Serializer.WriteString cliente.paterno
                Serializer.EndElement
                Serializer.StartElement "materno"
                    Serializer.WriteString cliente.materno
                Serializer.EndElement
                Serializer.StartElement "razonsocial"
                    Serializer.WriteString cliente.razonsocial
                Serializer.EndElement
                Serializer.StartElement "calle"
                    Serializer.WriteString cliente.calle
                Serializer.EndElement
                Serializer.StartElement "numero"
                    Serializer.WriteString cliente.numero
                Serializer.EndElement
                Serializer.StartElement "colonia"
                    Serializer.WriteString cliente.colonia
                Serializer.EndElement
                Serializer.StartElement "municipio"
                    Serializer.WriteString cliente.municipio
                Serializer.EndElement
                Serializer.StartElement "cp"
                    Serializer.WriteString cliente.cp
                Serializer.EndElement
                Serializer.WriteXml "<telefonos>"
                For i = 1 To UBound(arrayTelefonos)
                    Serializer.WriteXml "<telefono>"
                    Serializer.WriteXml "<tipodetelefono>" + arrayTelefonos(i, 1) + "</tipodetelefono>"
                    Serializer.WriteXml "<lada>" + arrayTelefonos(i, 2) + "</lada>"
                    Serializer.WriteXml "<numero>" + arrayTelefonos(i, 3) + "</numero>"
                    Serializer.WriteXml "</telefono>"
                Next i
                Serializer.WriteXml "</telefonos>"
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream


    If Not Reader.Fault Is Nothing Then
        AltaCliente = Reader.FaultString.Text
    Else
        AltaCliente = Reader.RpcResult.Text
    End If
End Function
```

AltaInventario

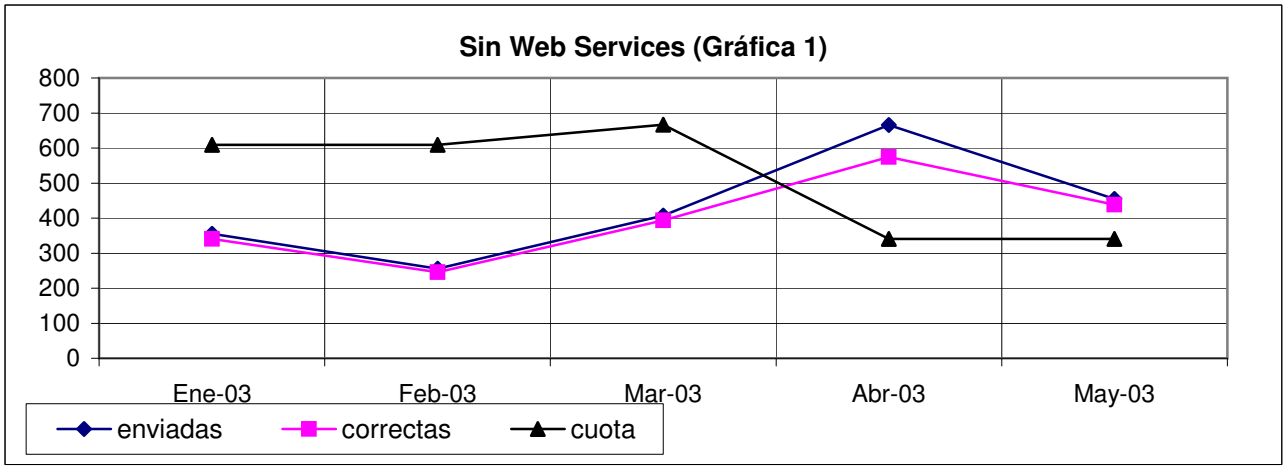
```
Function AltaInventario(ByVal agenciaid As String, password As String, _
                        ByRef arraySerie)
    Dim Serializer As SoapSerializer30
    Dim Reader As SoapReader30
    Dim ResultElm As IXMLDOMElement
    Dim FaultElm As IXMLDOMElement
    Dim Connector As ISoapConnector
    Dim i As Integer, cadena As String
    Set Connector = New HttpConnector30
    Connector.Property("EndPointURL") = END_POINT_URL
    Connector.Connect

    Connector.Property("SoapAction") = "http://www.unam.mx/AltaInventario"
    Connector.BeginMessage

    Set Serializer = New SoapSerializer30
    Serializer.Init Connector.InputStream
    Serializer.StartEnvelope
        Serializer.StartBody
            Serializer.StartElement "AltaInventario"
                Serializer.StartElement "agenciaid"
```

```
                    Serializer.WriteString agenciaid
                Serializer.EndElement
                Serializer.StartElement "password"
                    Serializer.WriteString password
                Serializer.EndElement
                Serializer.WriteXml "<inventario>"
                For i = 1 To UBound(arraySerie)
                    Serializer.WriteXml "<serieVehiculo>" + arraySerie(i) + "</serieVehiculo>"
                Next i
                Serializer.WriteXml "</inventario>"
            Serializer.EndElement
        Serializer.EndBody
    Serializer.EndEnvelope
    Connector.EndMessage

    Set Reader = New SoapReader30
    Reader.Load Connector.OutputStream


    If Not Reader.Fault Is Nothing Then
        AltaInventario = Reader.FaultString.Text
    Else
        AltaInventario = Reader.RpcResult.Text
    End If

End Function
```

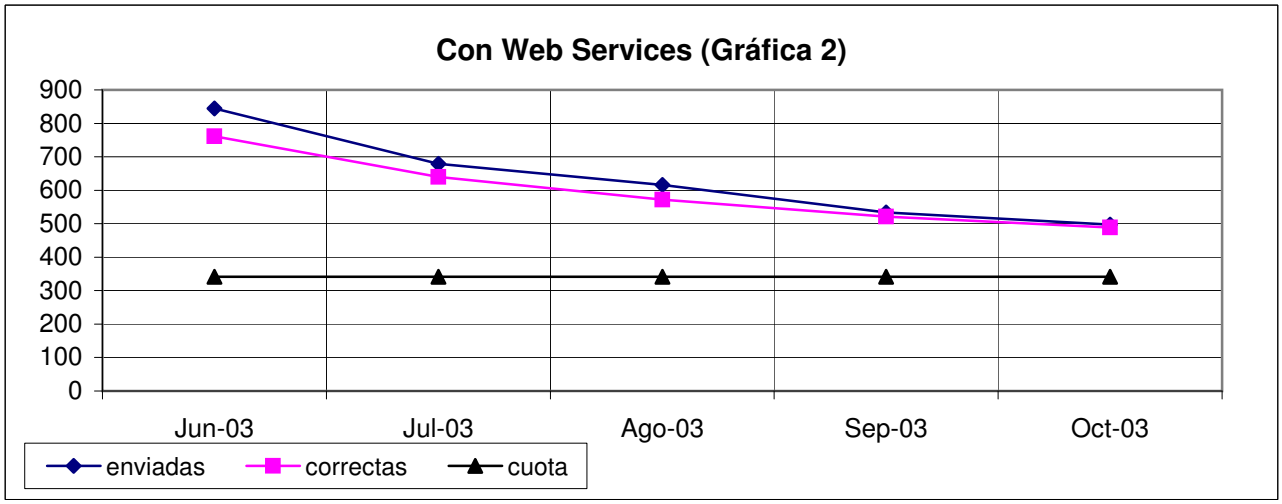
Anexo 15. Gráficas de Resultados

Caso I. Avenida Universidad

¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
no	Enero-2003	355	341	609	96.06	55.99
no	Febrero-2003	256	246	609	96.09	40.39
no	Marzo-2003	407	394	667	96.81	59.07
no	Abril-2003	666	574	341	86.19	168.33
no	Mayo-2003	455	438	341	96.26	128.45
	TOTAL	2139	1993	2567	93.17	77.64
	PROMEDIO	427.8	398.6	513.4	93.17	77.64

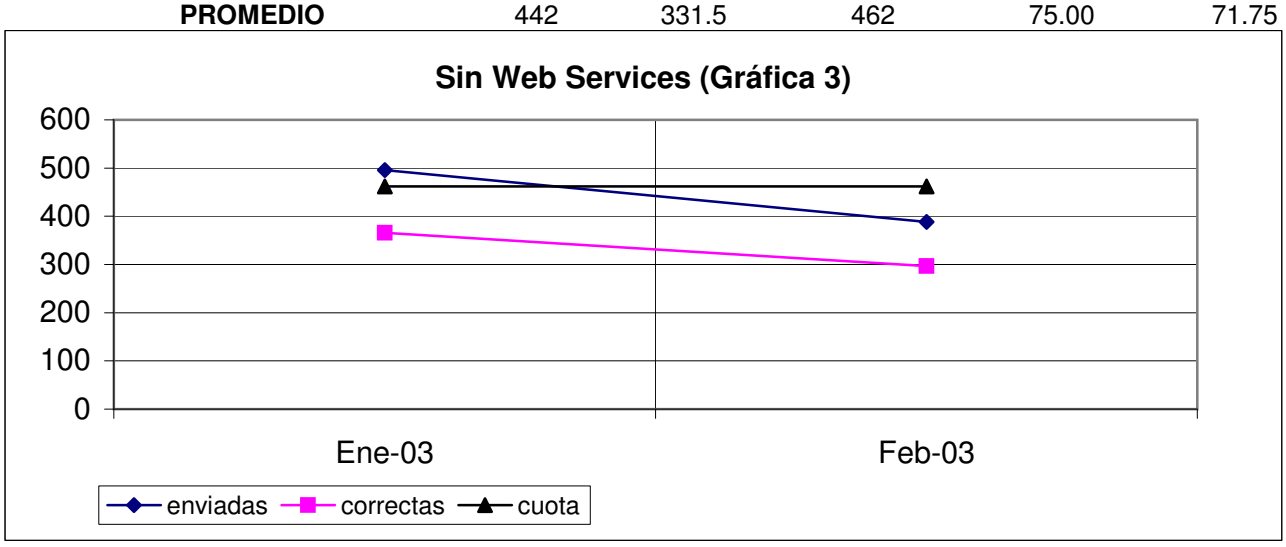


¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
si	Junio-2003	845	761	341	90.06	223.17
si	Julio-2003	679	640	341	94.26	187.68
si	Agosto-2003	616	572	341	92.86	167.74
si	Septiembre-2003	534	521	341	97.57	152.79
si	Octubre-2003	497	489	341	98.39	143.40
	TOTAL	3171	2983	1705	94.07	174.96
	PROMEDIO	634.2	596.6	341	94.07	174.96

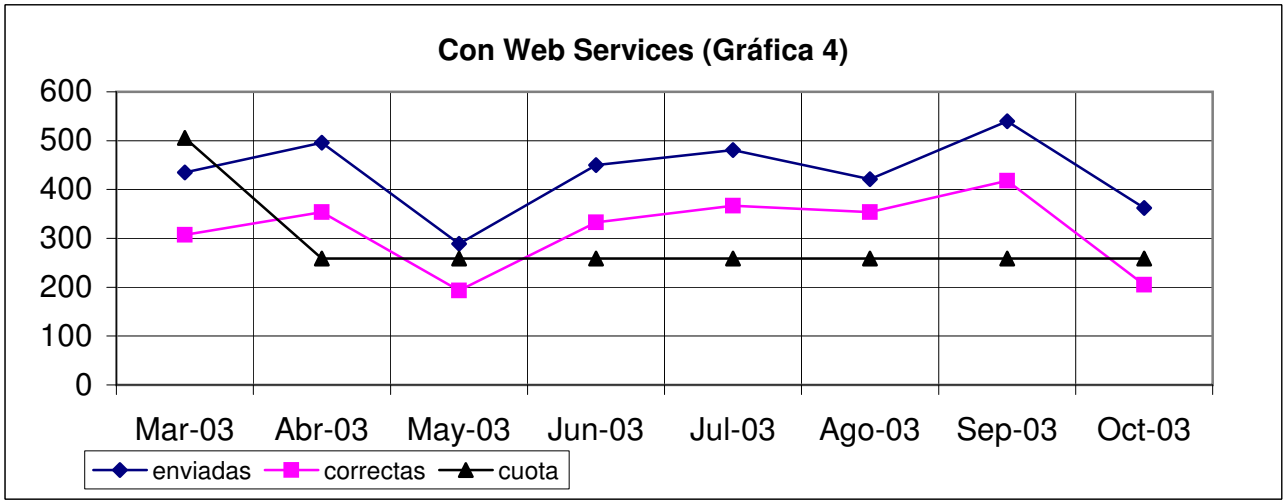


Caso II. Calzada de la Viga

¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
no	Enero-2003	496	366	462	73.79	79.22
no	Febrero-2003	388	297	462	76.55	64.29
	TOTAL	884	663	924	75.00	71.75
	PROMEDIO	442	331.5	462	75.00	71.75

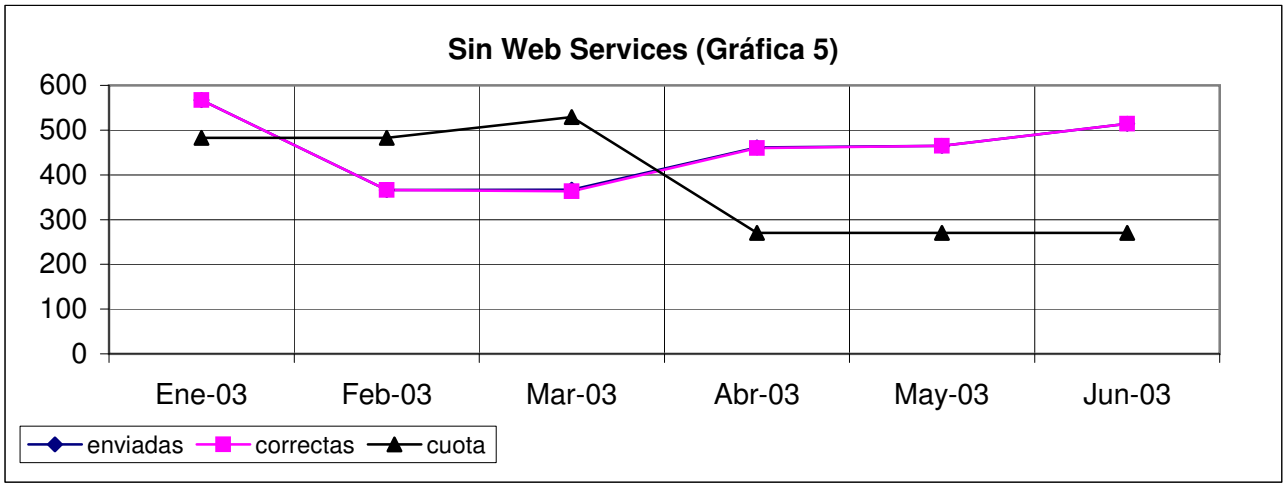


¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
	Marzo-2003	435	307	506	70.57	60.67
	Abril-2003	496	354	259	71.37	136.68
	Mayo-2003	289	193	259	66.78	74.52
si	Junio-2003	450	333	259	74.00	128.57
si	Julio-2003	481	367	259	76.30	141.70
si	Agosto-2003	421	354	259	84.09	136.68
si	Septiembre-2003	540	418	259	77.41	161.39
si	Octubre-2003	362	205	259	56.63	79.15
	TOTAL	3474	2531	2319	72.86	109.14
	PROMEDIO	434.25	316.375	289.875	72.86	109.14

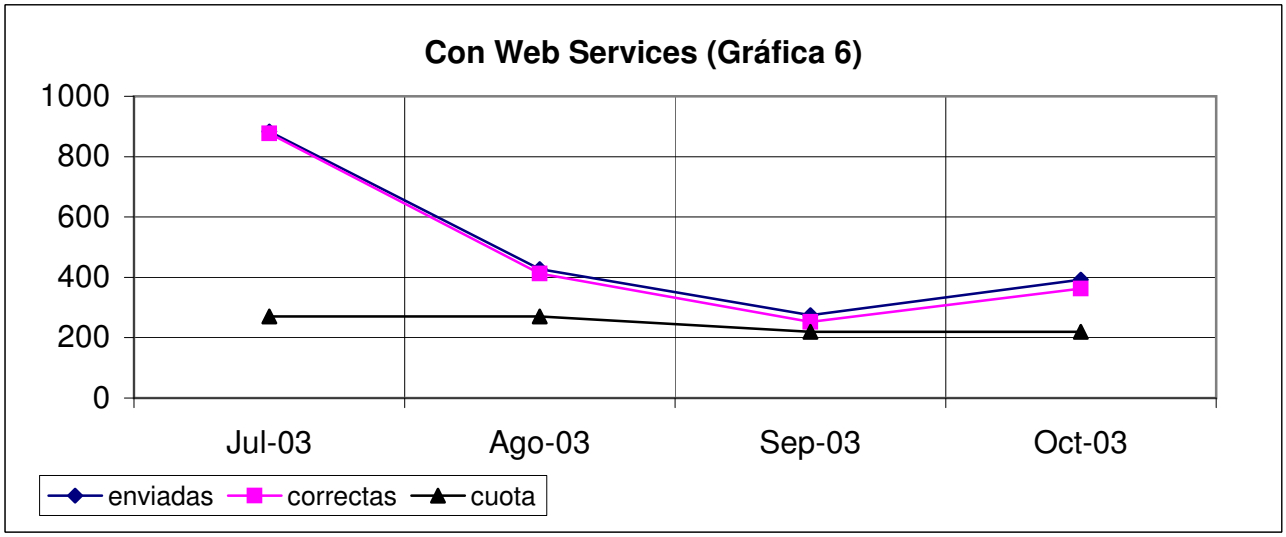


Caso III. Tlalpan

¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
no	Enero-2003	567	567	483	100.00	117.39
no	Febrero-2003	366	366	483	100.00	75.78
no	Marzo-2003	367	363	529	98.91	68.62
no	Abril-2003	461	460	270	99.78	170.37
no	Mayo-2003	465	465	270	100.00	172.22
no	Junio-2003	514	514	270	100.00	190.37
	TOTAL	2740	2735	2305	99.82	118.66
	PROMEDIO	456.67	455.83	384.17	99.82	118.66

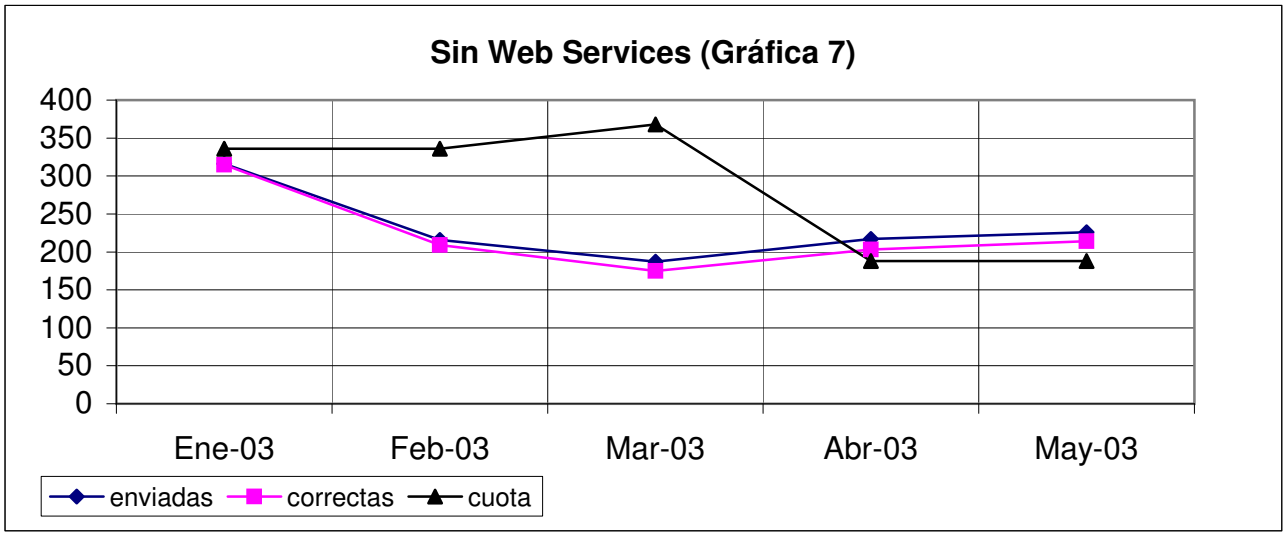


¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
si	Julio-2003	883	877	270	99.32	324.81
si	Agosto-2003	427	413	270	96.72	152.96
si	Septiembre-2003	275	252	220	91.64	114.55
si	Octubre-2003	392	363	220	92.60	165.00
	TOTAL	1977	1905	980	96.36	194.39
	PROMEDIO	494.25	476.25	245	96.36	194.39

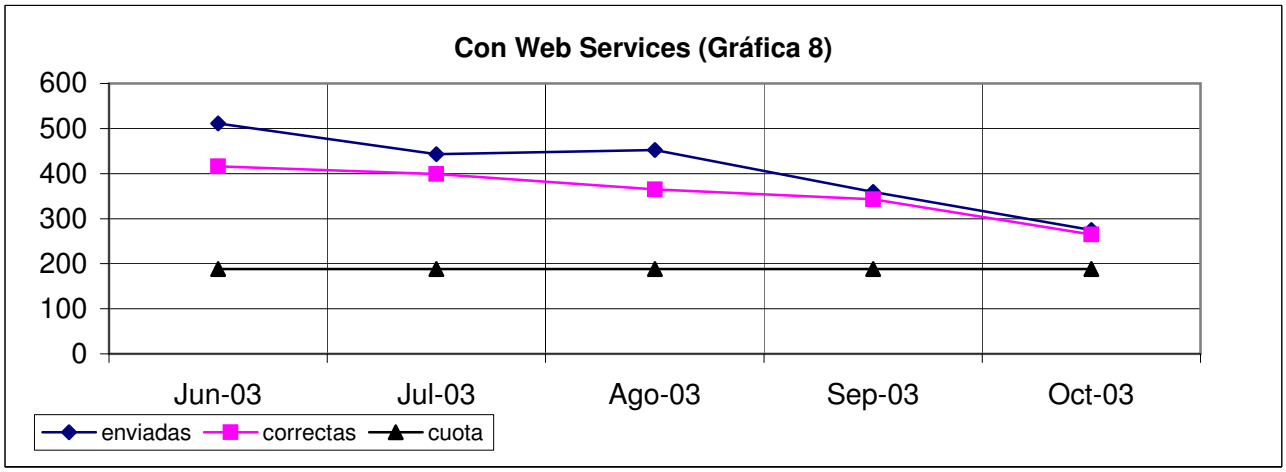


Caso IV. Carretera México Toluca

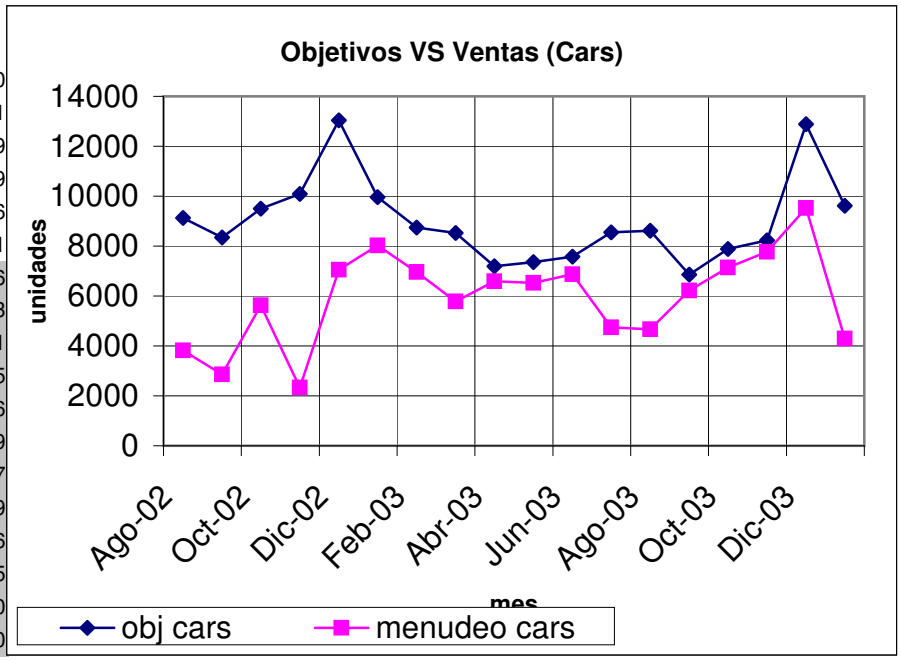
¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
no	Enero-2003	316	315	336	99.68	93.75
no	Febrero-2003	216	209	336	96.76	62.20
no	Marzo-2003	187	175	368	93.58	47.55
no	Abril-2003	217	203	188	93.55	107.98
no	Mayo-2003	226	214	188	94.69	113.83
	TOTAL	1162	1116	1416	96.04	78.81
	PROMEDIO	232.4	223.2	283.2	96.04	78.81



¿usa WS?	mes	enviadas	correctas	cuota	% de correctas	% de objetivos
si	Junio-2003	511	416	188	81.41	221.28
si	Julio-2003	443	399	188	90.07	212.23
si	Agosto-2003	452	365	188	80.75	194.15
si	Septiembre-2003	359	343	188	95.54	182.45
si	Octubre-2003	275	265	188	96.36	140.96
	TOTAL	2040	1788	940	87.65	190.21
	PROMEDIO	408	357.6	188	87.65	190.21

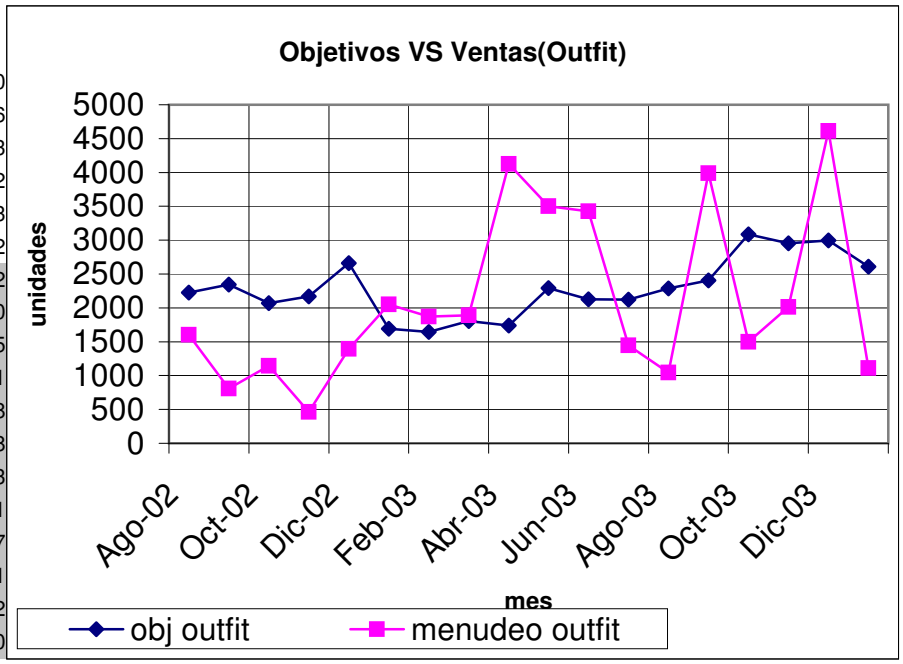


mes	obj cars	menudeo cars
Ago-02	9131	3820
Sep-02	8344	2861
Oct-02	9499	5629
Nov-02	10085	2339
Dic-02	13037	7056
Ene-03	9967	8031
Feb-03	8739	6966
Mar-03	8525	5783
Abr-03	7191	6591
May-03	7355	6525
Jun-03	7574	6866
Jul-03	8557	4749
Ago-03	8615	4667
Sep-03	6854	6219
Oct-03	7880	7146
Nov-03	8236	7765
Dic-03	12887	9530
Ene-04	9612	4300



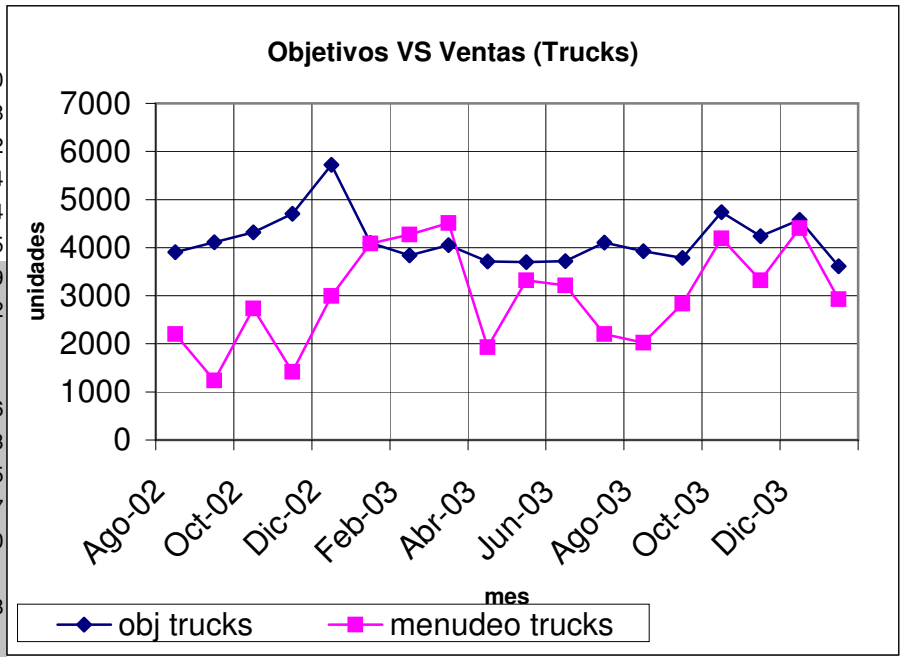
Gráfica 9

mes	obj outfit	menudeo outfit
Ago-02	2226	1600
Sep-02	2346	806
Oct-02	2069	1143
Nov-02	2169	462
Dic-02	2663	1393
Ene-03	1694	2052
Feb-03	1643	1872
Mar-03	1803	1890
Abr-03	1741	4125
May-03	2290	3501
Jun-03	2127	3428
Jul-03	2124	1448
Ago-03	2289	1043
Sep-03	2406	3991
Oct-03	3088	1497
Nov-03	2952	2011
Dic-03	2995	4612
Ene-04	2609	1110



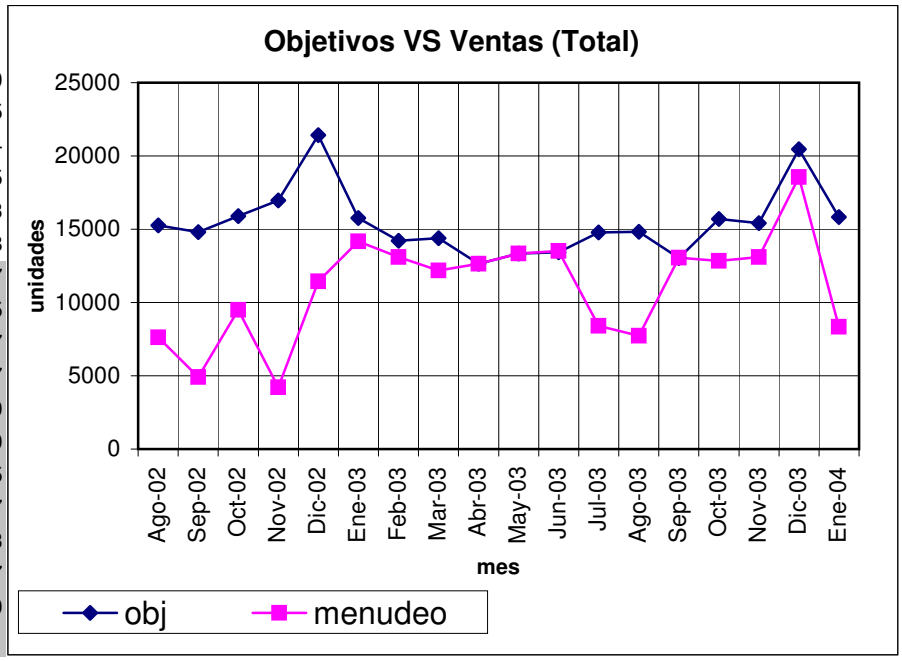
Gráfica 10

mes	obj trucks	menudeo trucks
Ago-02	3905	2200
Sep-02	4110	1238
Oct-02	4320	2732
Nov-02	4704	1414
Dic-02	5724	2994
Ene-03	4102	4085
Feb-03	3840	4269
Mar-03	4050	4512
Abr-03	3716	1931
May-03	3702	3321
Jun-03	3720	3216
Jul-03	4104	2203
Ago-03	3923	2025
Sep-03	3784	2837
Oct-03	4736	4190
Nov-03	4236	3321
Dic-03	4575	4408
Ene-04	3611	2931



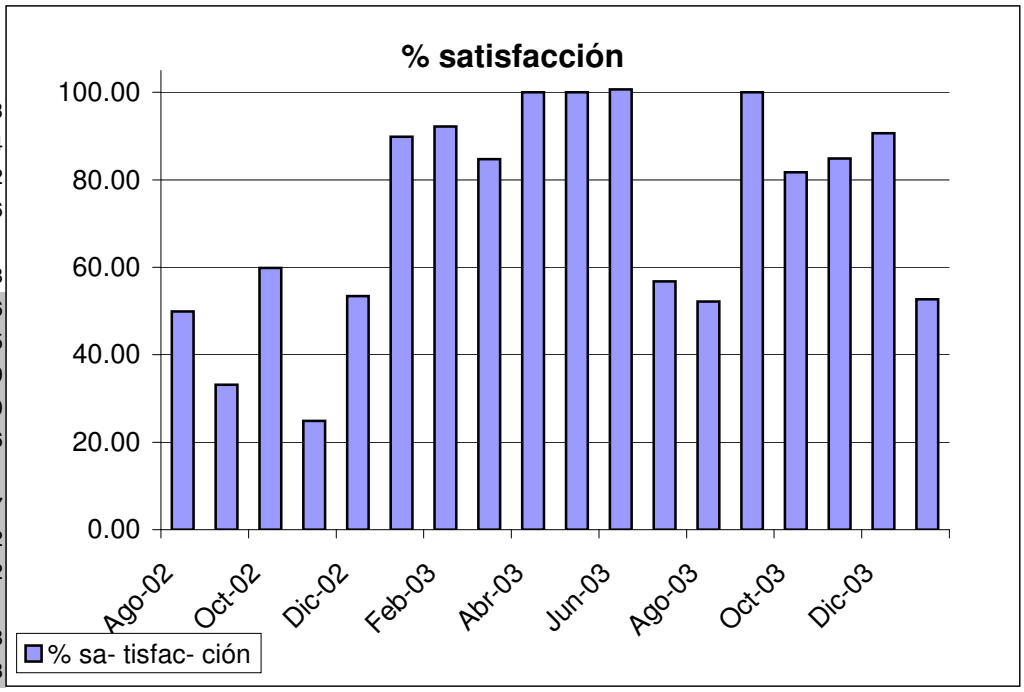
Gráfica 11

mes	obj	menudeo
Ago-02	15262	7620
Sep-02	14800	4905
Oct-02	15888	9504
Nov-02	16958	4215
Dic-02	21424	11443
Ene-03	15763	14168
Feb-03	14222	13107
Mar-03	14378	12185
Abr-03	12648	12647
May-03	13347	13347
Jun-03	13421	13510
Jul-03	14785	8400
Ago-03	14827	7735
Sep-03	13044	13047
Oct-03	15704	12833
Nov-03	15424	13097
Dic-03	20457	18550
Ene-04	15832	8341



Gráfica 12

mes	% satisfacción
Ago-02	49.93
Sep-02	33.14
Oct-02	59.82
Nov-02	24.86
Dic-02	53.41
Ene-03	89.88
Feb-03	92.16
Mar-03	84.75
Abr-03	99.99
May-03	100.00
Jun-03	100.66
Jul-03	56.81
Ago-03	52.17
Sep-03	100.02
Oct-03	81.72
Nov-03	84.91
Dic-03	90.68
Ene-04	52.68



Gráfica 13

Bibliografía

Publicaciones Impresas

VINOSKY, Steve. HENNING, Michi. Advanced CORBA Programming With C++. Addison Wesley professional computing series, Massachusetts, 1999. 1083 P.

GLASS, Graham. Web Services: Building Blocks for Distributed Systems (With CD-ROM) . Prentice Hall. USA, 2001. 272 p.

MAGELANG INSTITUTE. Introduction to CORBA. En: Tutorials & Code Camps. Sun Microsystems. USA, 2004

SHORT, Scott. Building XML Web Services For The Microsoft .NET Platform. Microsoft Press, Washington, 2002. 426 P.

MICROSOFT CORPORATION. "Overview of XML Documents" En: Building XML Based web Application. Microsoft Official Curriculum Course 1905C. México, Microsoft Press 2002. 10 Modulos.

MICROSOFT CORPORATION. "Introduction to SOAP and XML Web Services" En: Building XML Based web Application. Microsoft Official Curriculum Course 1905C. México, Microsoft Press 2002. 10 Modulos.

MICROSOFT CORPORATION. "Validating XML Using Schemas" En: Building XML Based web Application. Microsoft Official Curriculum Course 1905C. México, Microsoft Press 2002. 10 Modulos.

MICROSOFT CORPORATION. "The Need for XML Web Services" En:Developing XML Web Services using Microsoft ASP.NET. Microsoft Official Curriculum Course 2524B. México, Microsoft Press 2002. 9 Modulos.

MICROSOFT CORPORATION. "The Underlying Technologies of XML Web Services" En:Developing XML Web Services using Microsoft ASP.NET. Microsoft Official Curriculum Course 2524B. México, Microsoft Press 2002. 9 Módulos.

SICAP. Arquitectura de Aplicaciones Empresariales en Java. Soluciones Integrales de Capacitación, México, 2003. 200 p.

HALL, Marty. Servlets y JavaServer Pages. Guía Práctica. Pearson Educación, México, 2001. 608 p.

CLEMENTS, Nick. DAUX, Patrice. WILLIAMS, Gary. Java Programming Volumen 1. Oracle University, USA, 2000. 728 p. (2 vol. en la colección)

BAENA, Guillermina. MONTERO, Sergio. Tesis en 30 Días. Editores Mexicanos Unidos, México, 2003. 104 p.

FUENTES Zenón, Arturo. "El enfoque de Sistemas en la Solución de Problemas la elaboración del modelo conceptual". En: Cuadernos de planeación y sistemas Volumen 4. UNAM, Ingeniería, DEPFI. Ciudad Universitaria, México, 1995. 41 p. (9 vol. en la colección)

LUCENA Cayuela, Núria., et. al., Diccionario Enciclopédico 2003. Ediciones Larousse, México, 2003.1826 p.

WIDENIUS, Michael Monty. AXMARK, David. TOLONEN Jani, et. al., MySQL Reference Manual. MySQL AB, USA, 2003. 1025 p.

ARMSTRONG, Eric. BALL, Jennifer. BODOFF, Stephanie. et.al. The Java Web Services Tutorial. Sun Microsystems, USA, 2003. 1205 p.

Sun Microsystems. CORBA Technology and the JavaTM 2 Platform, Standard Edition. USA, CA. 2001 En: Java Documentation :\j2sdk1.4.2\docs\guide\corba\index.html

Uniform Resource Locators en Internet

Sitio de Graham Glass
http://www.perfectxml.com/nsc.asp?isbn=0130662569

Sitio de MageLang Institute
http://java.sun.com/developer/onlineTraining/corba/corba.html

Sitio de Sun Microsystems (Java 2 Standard Edition, Java 2 Enterprise Edition, Java Web Services Developer´s Package)
http://java.sun.com/

Sitio de Motor de Bases de datos MySQL AB
http://www.mysql.com

Sitio de tutorial de protocolo HTTP
`http://www.garshol.priv.no/download/text/http-tut.html`

Sitio de RMI
`http://www.comp.hkbu.edu.hk/~kchui/rmi/rmi.doc`

Sitio D VInt Productions especializado en XML Schemas
`http://www.xml.dvint.com`

Sitio de la W3C
`http://www.w3c.com`

Sitio de Pearson Education
`http://www.pearsonedlatino.com/ServletsJSP`